# NAVAL POSTGRADUATE SCHOOL
## Monterey, California

## THESIS

ON THE ROLE OF THE WORLD WIDE WEB
AND
WEB TECHNOLOGY IN EDUCATIONAL COURSEWARE

by

James W. Hester, Jr.
&
Richard C. Moormann

March 1997

Thesis Advisor:                                      Dennis  Volpano

# 19971124 027

# REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

| 1. AGENCY USE ONLY *(Leave blank)* | 2. REPORT DATE <br> March 1997 | 3. REPORT TYPE AND DATES COVERED <br> Master's Thesis |
|---|---|---|

| 4. TITLE AND SUBTITLE **ON THE ROLE OF THE WORLD WIDE WEB AND WEB TECHNOLOGY IN EDUCATIONAL COURSEWARE** | 5. FUNDING NUMBERS |
|---|---|
| 6. AUTHOR(S) <br> Hester, James W., Jr. <br> Moormann, Richard C. | |

| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) <br> Naval Postgraduate School <br> Monterey CA 93943-5000 | 8. PERFORMING ORGANIZATION REPORT NUMBER |
|---|---|
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER |

**11. SUPPLEMENTARY NOTES**

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

| 12a. DISTRIBUTION/AVAILABILITY STATEMENT <br> Approved for public release; distribution is unlimited. | 12b. DISTRIBUTION CODE |
|---|---|

**13. ABSTRACT** *(maximum 200 words)*

There are many types of computerized training systems available today ranging from text-based "quizzers" to interactive multimedia "edutainment". However, each system is proprietary to the platform for which the binary executable is compiled. Additionally, when the information in the training material changes, a new copy must be created, distributed and installed before it is available to the end user.

This thesis explores the use of the Java programming language as a fundamental element in the creation of interactive courseware deployable through the World Wide Web. Java is used to add interactive, executable content to Web pages in the form of simulations and complex demonstrations of educational concepts.

Complete online materials were developed in support of the initial offering of CS2973, a Java programming course. Following that success, a prototype interactive online exam system, using a Java applet and file server, was developed. This prototype foreshadows a complete virtual classroom environment supported by a Courseware Creation Interface. Both of these have the distinct advantage of being cross-platform by virtue of being created in the Java programming language, thus usable on a majority of operating systems and platforms through Java-enhanced Web browsers.

| 14. SUBJECT TERMS <br> Java, HTML, Virtual Classrooms, Object Oriented Programming, Internet Education, Online Training, Online Education, World Wide Web, Net Training, Courseware | 15. NUMBER OF PAGES 253 |
|---|---|
| | 16. PRICE CODE |

| 17. SECURITY CLASSIFICATION OF REPORT <br> Unclassified | 18. SECURITY CLASSIFICATION OF THIS PAGE <br> Unclassified | 19. SECURITY CLASSIFICATION OF ABSTRACT <br> Unclassified | 20. LIMITATION OF ABSTRACT <br> UL |
|---|---|---|---|

# ON THE ROLE OF THE WORLD WIDE WEB
## AND
# WEB TECHNOLOGY IN EDUCATIONAL COURSEWARE

James W. Hester, Jr,
Captain, United States Army
B.S., University of Kentucky, 1986
and
Richard C. Moormann,
Lieutenant, United States Navy
B.S., University of Missouri-Columbia, 1988

Submitted in partial fulfillment
of the requirements for the degree of

## MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

## NAVAL POSTGRADUATE SCHOOL
### March 1997

Authors: _____

James W. Hester, Jr.

_____

Richard C. Moormann

Approved by: _____

Dennis Volpano, Thesis Advisor

_____

Cynthia Irvine, Second Reader

_____

Ted Lewis, Chairman

iii

# ABSTRACT

There are many types of computerized training systems available today ranging from text-based "quizzers" to interactive multimedia "edutainment". However, each system is proprietary to the platform for which the binary executable is compiled. Additionally, when the information in the training material changes, a new copy must be created, distributed and installed before it is available to the end user.

This thesis explores the use of the Java programming language as a fundamental element in the creation of interactive courseware deployable through the World Wide Web. Java is used to add interactive, executable content to Web pages in the form of simulations and complex demonstrations of educational concepts.

Complete online materials were developed in support of the initial offering of CS2973, a Java programming course. Following that success, a prototype interactive online exam system, using a Java applet and file server, was developed. This prototype foreshadows a complete virtual classroom environment supported by a Courseware Creation Interface. Both of these have the distinct advantage of being cross-platform by virtue of being created in the Java programming language, thus usable on a majority of operating systems and platforms through Java-enhanced Web browsers.

# TABLE OF CONTENTS

# LIST OF FIGURES

# ACKNOWLEDGEMENT

# I. INTRODUCTION

Traditional teaching materials typically include lecture notes and other support in the form of video, audio and software. These materials have traditionally taken the form of handouts or overhead slides. As a result, once these kinds of course materials are produced, they are often duplicated and require much effort to modify. But suppose there were an improved and integrated way to view all of these different types of materials, and moreover, that this could be done with just the click of a mouse from anywhere on the planet! With such technology, the opportunity arises for truly reusable courseware, on a global scale, where by "courseware" we mean the electronic representation of instructional materials. Such technology now exists by virtue of the World Wide Web (WWW).

## A.   THE WORLD WIDE WEB

The World Wide Web, was initially developed at CERN, a particle physics laboratory in Geneva Switzerland. The initial work began in 1989 and centered on the development of the HyperText Transmission Protocol (HTTP), which is a network protocol for requesting and transmitting Web files and documents that both Web servers and browsers can understand. Through the early 90s, CERN created tools to make the Web what it is today. They developed a Web server, as well as a text-based web browser and eventually made the server and browser software freely available on the Internet. [NSM95]

Tim Berners-Lee, initial creator of the World Wide Web project, describes the Web as follows:

> The WWW project merges the techniques of information retrieval and hypertext to make an easy but powerful global information system. The WWW world consists of documents, and links. Indexes are special documents which, rather than being read, may be searched. The result of such a search is another (virtual) document containing links to the documents found. The Web contains documents in many formats. Those documents which are hypertext, (real or virtual) contain links to other documents, or places within documents. All documents, whether real, virtual or indexes, look similar to the reader and are contained within the same addressing scheme. To follow a link, a reader clicks with a mouse (or types in a number if he or she has no mouse). To search an index, a reader gives keywords (or other search criteria). These are the only operations necessary to access the entire world of data. [NSM95]

Perhaps the main reason for the rapid acceptance and growth of the Web was the work done at the National Center for Supercomputer Applications (NCSA) at the University of Illinois in Urbana-Champaign. Their software development group created a graphical Web browser called Mosaic. In September 1993, they released versions of Mosaic for Microsoft Windows running on PCs, Apple Macintoshes and Unix computers running X Windows. Each of the versions handled files in a very similar manner with images and text interspersed in the same document, allowing organizations to create visually exciting documents that could be viewed in very similar formats on the three main types of computer in use at that time. [NSM95]

Many members of the team who developed the original versions of Mosaic now work for Netscape Communications Corporation, a company which has developed the Netscape Web browser. It is estimated that Netscape accounted for around 70 percent of all Web browsers in use in May 1995. Following the highly successful launch of Netscape

Communications, Microsoft recognized the commercial potential of the WWW and quickly developed the Internet Explorer Web browser. Many other companies have also launched a range of Internet browsers and servers, further intensifying competition in the Internet marketplace. [NSM95]

## B.    EDUCATION AND THE ROLE OF THE WEB

Until now, use of the Web for instruction has involved primarily the use of static content (text and images). Though somewhat limited, static content on the Web still has many advantages. Some of the advantages are that Web documents are available to anyone on the network running a Web browser. This includes remote Internet sites accessible by modem. Web documents also integrate easily with other network services like email and network newsgroups. They can be developed and shared by many institutions and companies for automated distance learning. Web documents can also be viewed by browsers running on many different kinds of platforms from Macintosh to Unix to PC compatibles. Another feature of Web documents is the ability to have hypertext links embedded with in Web documents. The hypertext links allow the Web page designer to link related Web pages to other Web pages. This allows users who require further detail on a topic to merely click on the hypertext link to move to related Web pages. And finally, many vendors offer a wide variety of Web document generation tools that greatly simplify building Web documents.

In this thesis, we investigate the role of the World Wide Web and technology, in educational courseware. Use of the Web for this purpose is facilitated by recent Web

programming languages like Sun's Java and Microsoft's Active X, and scripting languages like Netscape's JavaScript and Microsoft's Visual Basic Script. These tools, coupled with powerful Web browsers and enhanced server Applications Programming Interfaces (API's), make it possible to move far beyond the traditional "static" approaches to Web content that were popular just a short time ago. They provide a framework within which one can develop very flexible and interactive aids for online instruction and training. Within the simplicity of a Web browser exists the capacity to display textual data, play audio streams, display animated graphical representations of data (e.g. weather analysis) and to engage the user in interactive learning with simulations of any level of complexity.

## C. LEVELS OF WEB INTERACTION

We classify Web courseware roughly according to the level of interaction it provides. We consider the lowest level of interaction to be that of "static content" or static Web pages. This is essentially hypertext, implemented in Hypertext Markup Language (HTML) which is a subset of Standard General Markup Language (SGML). It is characterized by hypertext links consisting of Uniform Resource Locators (URL's). Through implementations of the simple GET, PUT and POST commands of the Hypertext Transport Protocol (HTTP), servers can provide the byte streams for images, audio or text, which we consider static.

At higher levels of interaction, we find "active content" in Web pages. Examples include the HTTP features of "server-side includes" and Common Gateway Interface (CGI) scripts written in Perl and C. With this technology it is possible to fill out forms in Web pages and store their contents on a server. CGI scripts go beyond the static model of

a client issuing one HTML request after another. Instead, CGI scripts allow the HTTP server to send documents depending on the client's request. Although useful for online forms and database queries, CGI scripts are much too awkward for serious courseware development. Since CGI scripts execute on the server, they can lead to poor performance when there are many requests to the server.

Opportunities for high level interaction came in 1995 with the introduction of Sun's HotJava Web browser. Unlike its predecessor, NCSA Mosaic, HotJava supported "active content" in Web pages called Java Applets. Active content is basically executable code. Though not originally targeted for the Web, Java turned out to be extraordinarily useful in this setting. Applets gave programmers a way to bring life to Web pages through animation and simulations. The seeds were sown for exciting new ways to conduct research and to educate using the Web. Not surprisingly, active content offers the most potential for developing effective online courseware.

Active content may take the form of an inline executable script or a procedure call, via some sort of tag in the markup language, which runs within a "container" (or runtime system) for the code. Our examples use the Netscape Navigator (v2.02) as the container in which Java bytecode is executed. However, the techniques we describe are really language and container independent. We suspect, although we did not attempt it, that our techniques could also be applied to Active X and Microsoft Internet Explorer.

## D.  THESIS ORGANIZATION

This thesis illustrates some of the ways Java and active content can be used to develop courseware that, contains a combination of hypertext and executable content in documents.  Some of the techniques described here were used to build courseware for the Naval Postgraduate School course, CS2973, Object-oriented Programming for the Internet with Java.  It was offered for the first time by the Computer Science Department in the Summer of 1996.  CS2973 courseware is entirely online, written primarily in HTML with some embedded Java.  We will also describe some techniques that go beyond what was employed in CS2973 to illustrate the great potential of Web technology for instruction.

Static content in Web pages is considered in Chapter II, followed by active content in Chapter III.  In Chapter IV, we discuss the use of Java in active courseware.  The merits of each are examined and examples are given.  Some examples are taken from the inaugural offering of CS2973.  In Chapter V, we discuss future work and in Chapter VI we discuss related work and draw some conclusions about the future of Web technology in education.

# II. STATIC CONTENT IN COURSEWARE

Static content can be a major component of effective courseware even though it is not active. This kind of content includes text, images and audio and thus provides the rudiments for an online textbook. Of course, the text or static content in courseware is hypertext, so in this sense, static content offers more than ordinary text. For example, it allows for self-describing pictures using audio files.

## A. HTML — THE GLUE OF STATIC CONTENT

The mechanism one uses for organizing static content into a useful format is the Hypertext Markup Language (HTML). In its simplest form, HTML "tags" are placed in plain text files to define the display characteristics of the marked text. Joint Photographic Experts Group (JPG) and Graphics Interchange Format (GIF) images may be embedded in the document for inline display with the text. Additionally, hyperlink tags, which point to resources external to the document, can be embedded easily. When the tags are interpreted by a Web browser, the text and images are formatted accordingly. Figure 2.1 lists the HTML code used to generate the Web page in Figure 2.2.

HTML is based on formatting with respect to components such as headings, paragraphs and lists  The general format for a HTML tag is

```
<tag_name>string of text</tag_name>.
```

The entire HTML document is delimited by the tags

```
<HTML> and </HTML>.
```

As shown in Figure 2.1, an HTML document contains two distinct parts, the head and the body. The head contains identifying information about the document, such as indexing information and the title, and is delimited by the tags

<HEAD> and </HEAD>.

The body contains everything displayed as part of the web page and is delimited by

<BODY> and </BODY>.

Among the more useful tags are the image and hyperlink anchor tags. An example of the HTML code required to embed an image in the page is the line

<IMG SRC="ballgrn.gif">.

The IMG tag indicates an image and the SRC value contains the filename of the image. A hyperlink may be embedded behind any text or image displayed in the document by using an anchor tag:

<a href="../course/index.html"> [Previous] </a>

which is a link to the URL of the index page. One can make an image a hyperlink by combining the previous two examples as in

```
<a href="../course/index.html"><IMG SRC="../graphics/globe.gif"
          ALT="GLOBE" align="right" ></a>.
```

This example also uses an alignment option and alternative text to be used if the browser does not display the image. HTML is discussed further in Appendix D.


## B.    USING STATIC CONTENT EFFECTIVELY

Here we provide some suggestions for using static web content effectively in courseware. They are the result of our experience in developing the CS2973 course content the Web. From this course, we learned some general guidelines for navigating Web documents and laying out Web pages effectively in courseware.

```
<HTML>
<HEAD>
<TITLE>Homework Info </TITLE>
</HEAD>
<BODY>
<a href="../course/index.html"> [Previous] </a>
<a href="hw1.html"> [Next] </a>
<a href="index.html"> [Index]</a>
<a href="../course/index.html"><IMG SRC="../graphics/globe.gif"
ALT="GLOBE" align="right" ></a>
<H1>CS2973 Homework Info</H1>
<IMG SRC="ballgrn.gif">
<A href="#setup">Required WWW Setup</a><BR>
<IMG SRC="ballgrn.gif">Assignments</a>
<BR>
<ul>
<LI><a href="hw1.html">Homework 1</a><BR>
<LI><a href="hw2.html">Homework 2</a><br>
<LI><a href="hw3.html">Homework 3</a><br>
<LI><a href="hw4.html">Homework 4</a><br>
<LI><a href="hw5.html">Homework 5</a><br>
</ul>
<p>
The CS2973 homework exercises constitute 60% of your final grade.<BR>
<B>Critical items are in bold.</b> Please adhere closely to these
instructions so that all
of the items are standardized.<p>
<h2><name="setup">Required WWW Setup</h2>
If you do not have one, <b>create a home page</b> for yourself on the
the NPS Web server. For more information look at the <a href=
"http://vislab-www.nps.navy.mil/YourOwn.html">VisLab's HowTo </a>.
 Ensure that your correct <B>email</B> address is listed on the page.<p>
<P>
this page is located at <BR><B>http://vislab-
www.nps.navy.mil/~java/course/hw.html</B><BR>
<A
HREF="mailto:java@nps.navy.mil(JAVAPage)"><i>java@nps.navy.mil</i></A><B
R>
Last Updated 8 JUL96<BR>
 <!--Copyright--> &#169 Naval Postgraduate School 1996<BR>
</BODY>
</HTML>
```

**Figure 2.1    HTML source code**

## C.    NAVIGATION

Whether a classroom of students is lead by an instructor or an individual is browsing

independently, the usual path through the material should be obvious and direct. On the other

hand, examining related topics or material should also be encouraged. This can be facilitated by

adopting the following navigation strategy.

First, plan a simple, hierarchical view of the course topics presented as an index page. The

course topics can be viewed at any level of granularity. For individual study, a student may go to

9

the topic of choice and skip the introductory material altogether. This allows students to customize their view of the course should they already be familiar with some of the material. It is not necessary to start from the beginning and continue to the end in a serial manner.

Second, define a clear path of instruction through the material. The path should consist of the course topics presented in logical order. Moving in either direction along the path should be accomplished by a single mouse click on the appropriate hyperlink. For example, the link [Previous] returns a user to the previous page, [Next] takes a user to the next page, and [Index] returns the user to the index page.

It should also be possible to explore a subject further at any time. Returning to the original point will be accomplished by using the "Back" feature of the browser. This allows students to navigate through related material, deviating from the main course path, without the problem of trying to find their way back to the original page. If a student wishes to back track to a previous page, then he or she merely "clicks on" the "Back" feature of the browser until the desired Web page is displayed.

## D.    PAGE LAYOUT

Continuity plays an important role in the acceptance of online courseware by users. Adopt a template to use as a model for every page in a course. Standard page headers should include navigational hyperlinks ([Previous], [Next], [Index]), the section number and the title of the section. The footer includes navigational hyperlinks, contact information for the page author, creation date, and the URL of the page being viewed. Standardizing the course page layout enhances the presentation and makes future modifications easier; see Figure 2.2.

**Navigation Links**
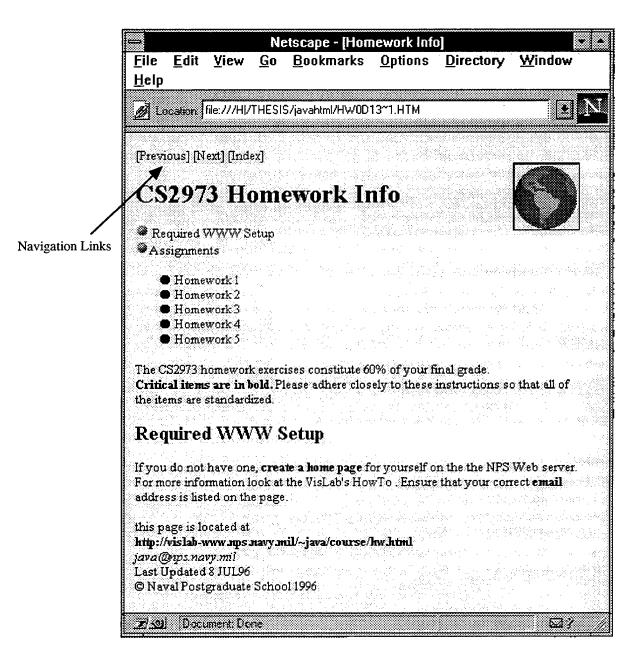


Figure 2.2 Standard page layout

# III. ACTIVE CONTENT VIA JAVA

For this thesis, we implemented our courseware in Java and executed it within the Netscape browser. Java is an Object-Oriented Programming Language that we chose because of its useful Application Program Interfaces (API) for platform-independent Graphical User Interfaces (GUIs) and network programming. Instead of generating binary executables, a Java compiler creates "bytecode" which execute via a Java interpreter on the native system. Java-enabled Web browsers execute bytecode as the HTML document is loaded. As such, Java enables the Web page to contain small programs with which the user may interact. Though there are numerous books on the Java language, in this chapter we will summarize the key features of Java needed for online courseware. They are the Java Abstract Windowing Toolkit (AWT), defined in the `java.awt` package, and Java networking tools defined in the `java.net` package.

## A.    GUI PROGRAMMING IN JAVA WITH THE AWT

The AWT is a platform-independent windowing toolkit supporting many of the basic needs of a GUI (e.g. windows, menu bars, buttons, etc). The toolkit is regarded as abstract because it serves as an interface between Java applications (and Applets) and native GUI libraries for a given operating system. Details of the underlying GUI library, which are platform-specific, are hidden from Java programs. Because the windowing sub-components, buttons, scrollbars, menus, etc., are actually provided by the native windowing system's "peer objects", there may be slight layout and display differences between platforms. For instance, the file dialogue display window on the Macintosh is different from that of the file dialogue display window on Microsoft Windows. In order

for the Java AWT to be portable, it must have an interface developed for each specific platform allowing the AWT to recognize the native system's peer objects. The Java AWT provides the "highest common factor" of functionality across a wide variety of native windowing toolkits. The AWT's methods for each platform are implemented by SunSoft, the makers of Java.

What does all that mean? In essence, a given piece of code written to display a set of buttons, labels or scrollbars for one platform, can be used on a completely different platform with no changes in the code because the Java program calls the peer for each system.

The following example is a simple "Hello World" applet. It does nothing more than print the string "Hello World". The Java sourcecode is shown in Figure 3.1. The `init()` method is invoked by the browser when the applet is initially loaded into the browser. The Label object calls the native peer of a label and displays the string "Hello World". The system font "Dialog" is specified with the characteristics of bold and 10 point size. This file is compiled to create the bytecode file `"hello.class"` and is called by the HTML file `"hello.html"`. The minimum HTML tags needed to display an applet are the code parameter, which is the name of the bytecode file, and the width and height of the applet in pixels. The HTML file in Figure 3.2 will produce the output of Figure 3.3. Though this is a simple example, using only the Label class of the AWT, it illustrates the basic relationship between the Java AWT and HTML. The AWT is described in more detail in Appendix E.

```
import java.awt.*;
public class hello extends java.applet.Applet {
    Label label1;
    public void init() {
        label1=new Label("Hello World");
        label1.setFont(new Font("Dialog",Font.BOLD,10));
        add(label1);
    }
}
```

**Figure 3.1  Java source code for hello.java.**

```
<HTML>
<HEAD>
<TITLE>My First Applet</TITLE>
</HEAD>
<BODY>
<P>
<APPLET code="hello.class" width=150 height=100>
</APPLET>
</BODY>
</HTML>
```

**Figure 3.2  hello.html source code.**
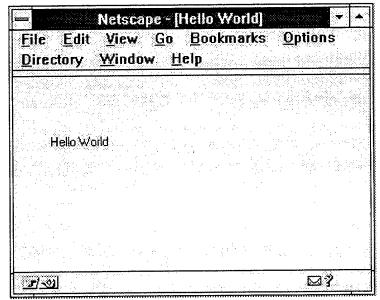


**Figure 3.3  Snapshot of hello.html.**

## B.    THE NETWORK API IN JAVA

Java provides a number of built-in networking capabilities that make it easy to

develop Web-based applications. Java's networking capabilities are provided via the

java.net package. The java.net package provides numerous powerful network

programming classes that simplify networking, making network programming with Java

flexible and easy to use (Figure 3.4). The java.net package supports both TCP/IP

(Transport Connection Protocol/Internet Protocol) and UDP (Unreliable Datagram

Protocol) protocol families.



```
                          ┌──────────────────┐
                          │ java net Package │
                          └──────────────────┘
   ┌──────────────────┐              ┌──────────────────┐
   │ Socketimpl       │──────┬───────│ ContentHandler   │
   └──────────────────┘      │       └──────────────────┘
   ┌──────────────────┐      │       ┌──────────────────┐
   │ URL              │──────┼───────│ DatagramPacket   │
   └──────────────────┘      │       └──────────────────┘
   ┌──────────────────┐      │       ┌──────────────────┐
   │ URLConnection    │──────┼───────│ DatagramSocket   │
   └──────────────────┘      │       └──────────────────┘
   ┌──────────────────┐      │       ┌──────────────────┐
   │ URLEncoder       │──────┼───────│ InetAddress      │
   └──────────────────┘      │       └──────────────────┘
   ┌──────────────────┐      │       ┌──────────────────┐
   │ URLStreamHanler  │──────┼───────│ ServerSocket     │
   └──────────────────┘      │       └──────────────────┘
   ┌──────────────────┐      │
   │ Socket           │──────┘
   └──────────────────┘
```
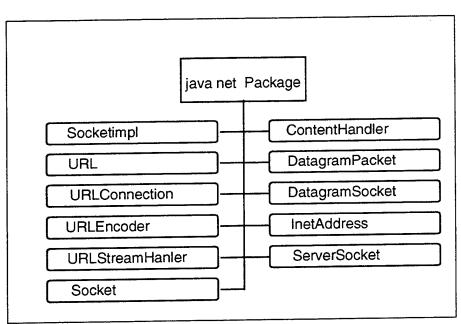
**Figure 3.4  java.net Package**

TCP/IP is used for reliable stream-based communication across the Internet. It

specifies the manner in which two processes running on different machines on the Internet

find each other, rendezvous, and transfer data. It makes sure data are transferred in the

16

correct sequence and without error. Additionally, TCP/IP uses a static addressing scheme where every machine on the Internet is assigned a unique, fixed IP address.

In contrast, UDP is used for unreliable datagram communication; also called "fire-and-forget" packet transmission across the network. UDP is fast, but the tradeoff is that the data is not guaranteed to reach its destination, and separate datagrams are not even guaranteed to reach their destination in the order in which they were sent. Communication with datagrams is useful when you want low-overhead communications of non-critical data and a stream model of communication is not necessary. In this case, the application must reorganize the data into the correct sequence.

Although the java.net package is organized into eleven classes that range from the simple URL networking class to the more complex DatagramPacket class, we will focus on the features useful in building courseware. These features allow the client applet to create socket connections to a server and to send and receive data.

The DatagramPacket class implements an unreliable datagram "packet" of data that may be sent or received over the network through a DatagramSocket.

The DatagramSocket class defines a socket that can receive and send unreliable DatagramPacket class implementations over the network.

The InetAddress class represents an Internet address, and is used when creating DatagramPacket or Socket objects. This class returns an array of bytes which represents the name, internet protocol address, local host name, and other information.

The ServerSocket class is used by servers to listen for socket connection requests from clients. The Socket class implements a socket for interprocess communica tion over the network. The constructor method creates the socket and connects it to the

17

specific host and port. TCP/IP sockets are used to implement reliable, bi-directional,
stream-based connections between a server and a client on the Internet. In Java, TCP/IP
sockets are created via the Socket class.

## 1. A Java Network Application — Course Server and Question Applet

Java provides socket-based communications that enable applications to view
network communications as if they were simple file input and output. A program can read
from a socket or write to a socket as simply as reading or writing with a file. The socket
class implements a reliable stream network connection. A common model for networking
is to have one or more clients send requests to a single server program. In Chapter IV we
describe a server application and a client applet which pass streams of data via socket
connection. The sourcecode will be used as an example here.

### a. Course Server

The Course Server uses the serverSocket class to accept connections
from clients. When a client connects to the port that a serverSocket is listening on, the
server allocates a new socket object, which is connected to a defined port, for the client
to communicate through (Figure 3.5). Since the Course Server is multithreaded, it then
returns to listening for other possible clients on the serverSocket (Figure 3.6). See
Appendix G for a complete description of the Course Server.

```
// Create a ServerSocket to listen for connections on;
public FS3(int port) {
        if (port == 0) port = DEFAULT_PORT;
        this.port = port;
        try { listen_socket = new ServerSocket(port); }
        catch (IOException e) {
          fail(e, "Exception creating server socket"); }
        System.out.println("FS3: listening on port "+port);
        this.start();
}
```

**Figure 3.5** serverSocket **listening on socket,** FS3.java **source code**

```
// The body of the server thread.  Loop forever,
// listening for and accepting connections from clients.
// For each connection, create a Connection object to
// handle communication through the new Socket.  Multiple Threads.
    public void run() {
        try {
            while(true) {
                Socket client_socket = listen_socket.accept();
                Connection c = new Connection(client_socket);
            }
        }
        catch (IOException e) { fail(e, "Exception while listening
for connections");
        }
}
```

**Figure 3.6** `ServerSocket` **listening, spawn new socket upon connection.**

### b.    Question Applet

The `QuestionApplet` class implements a client that communicates with the

Course Server. The `QuestionApplet` creates a `Socket` object, as shown in Figure 3.7, to

establish the connection to the Course Server. The `QuestionApplet` client creates a

DataInputStream to read lines from the Course Server until it encounters an "end-of-file"

character. When the "end-of-file" character is read, the client then closes the socket

connection as shown in Figure 3.8. See Appendix H for a complete description of the

QuestionApplet.

```
try {
    s = new Socket(this.getCodeBase().getHost(), PORT);
    out = new PrintStream(s.getOutputStream());
    out.println(score+"|"+lesson_file);
System.out.println("Score|Lesson_file="+score+"|"+lesson_file);
    }
catch (IOException e) {
    System.out.println("No Socket->"+e.toString());
    error("Socket Could Not Be Opened!");
    }
```

**Figure 3.7 QuestionApplet creating socket.**

```
try {
    //*** wait until it gets a stream from server
    for(;;) {
        line = in.readLine();   //** read in string line from server
            if (line != null) break;
                num_trys = (num_trys +1);
                //** build a dialog to ask whether to abort or continue
                if (num_trys > 80) {
                    error("Nothing coming in from server");
                }
    } //** for
} //** try
    catch (IOException e) {
            error("Error Reading Lesson From File!");
        } //** if
    finally {
        try {
            if (s != null) {
                s.close(); //** socket closed by Client
            }
        }
        catch(IOException e2) {
            error("Socket Could Not Be Closed!");
        }
    } //** finally
```

**Figure 3.8 QuestionApplet reads until no more data.**

The networking and AWT packages will be used throughout Chapter IV to

illustrate the usefulness of Java in creating active courseware via the Web.

20

# IV. ACTIVE CONTENT IN COURSEWARE

Using the basics of HTML discussed in the Chapter II, raw text can be formatted in a Web page to look more like a cleanly printed page than Courier-based ASCII text. Images may be added to support material described in the text. For the most part, these pages could be printed from the browser onto paper and used as notes or any other reference material. Many forms of previously-printed materials, from textbooks to user manuals, have been converted into HTML. The next step is to take advantage of active content to allow the user to interact with the instructional material.

Active content within Web pages is expressed in a programming language. In some cases, this may be a scripting language (JavaScript, VBScript, etc.) and in other cases a more general purpose language (Java, ActiveX). Depending on the kind of Web-based courseware being developed, one may choose among these different languages. Some offer the ability to modify local files on a client's machine while others do not. If this is not a design issue, then the choice may be determined by the desired features of a target browser. Both Netscape and Microsoft Web browsers support Java while ActiveX is proprietary only to Microsoft. The security model of the Netscape implementation of the Java interpreter does not allow full access to the client's operating system while ActiveX has full access.

To illustrate these concepts, we look at three examples with increasing levels of interaction. Though these examples are based on our experience in teaching a programming course, they could be about most any topic that has rules, illustrations and examples for students to follow. The first applet is an educational aid taken from CS2973. It helps to explain the concept of scope for `public`, `private` and `protected` visibility modifiers in the

21

Java programming language. The second applet helps to explain the use of Java arrays. It creates frames external to the browser and creates runtime objects based on the user input. Our final example illustrates truly active courseware. It involves an applet that connects to a dedicated server in order to retrieve questions for a user and then scores the results before sending the user the next lesson.

## A.    AN EXAMPLE OF ACTIVE CONTENT IN COURSEWARE

Our first example, which illustrates a limited form of interaction, is taken from a section of the online notes for the Java programming course CS2973. It is an applet that helps to explain the class, method and variable visibility modifiers used in the Java programming language.

The topic of packages and access protection can be quite confusing. An attempt to convey the rules of visibility is given via a static web page in Figure 4.1. One must refer to a language reference manual and then back to the web page in order to understand the rules. For example, to answer the second question "can f see i", depends on whether class c is declared `public` or `private` (Appendix F). We could introduce additional text that explains this or even include a table like that from Java in a Nutshell[Nutshell96: p 179]. But this forces a user to find the rule of interest among the many listed.

The embedded Java applet, shown in Figures 4.2-4.4, allows the reader to independently experiment with the various access modifiers and gain immediate feedback as to how access is affected. The allowable keywords to enter into the upper box are: `private`, `public`, `protected`, `private protected` or just left blank. Once a keyword is entered in the upper box, the answer to the question "can f see i ?" is revealed in the lower box. An

22

erroneous entry is rejected with an error message, as shown in Figure 4.5. In the enhanced

version, the reader has no doubt about the access rules since the results are displayed

immediately after the modifier is entered. The full text of the applet is listed in Appendix F.
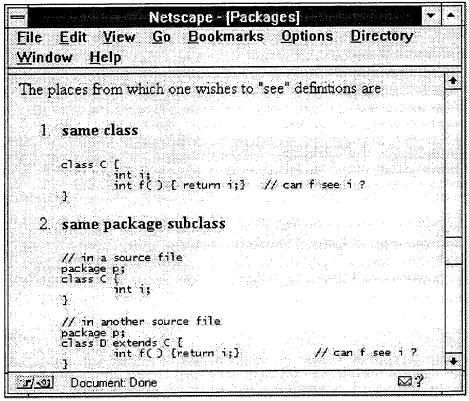


**Figure 4.1 Using static content to explain visibility modifiers.**

## B.  ANOTHER EXAMPLE OF ACTIVE CONTENT

Within the scope of CS2973, another useful example is the concept of an array

definition and allocation. Textually, one can describe an array as "a vector or object that stores

other objects or values in cells, referred to as elements" and then present an illustration

showing buckets to hold the values. With the Java applet embedded in the text, the reader may

additionally manipulate the graphical representation of an array by specifying the dimensions

and actually placing values in the cells. Furthermore, the example demonstrates how elements

in the array can be accessed. The values entered into the cells of the newly created elements

File   Edit   View   Go   Bookmarks   Options   Directory   Window   Help

2.  same package subclass

```
// A SOURCE FILE

package p;

┌──────────────┐
│              │class C {
└──────────────┘
    int i;

}

// IN ANOTHER SOURCE FILE

package p;

class D extends C {

    int f() {return i;}            // can f see i ?   ┌──────────┐
                                                     │          │
                                                     └──────────┘

}
```

**Figure 4.2  Using active content to explain visibility modifiers, initialized.**

2.  same package subclass

```
// A SOURCE FILE

package p;

┌──────────────┐
│ private      │class C {
└──────────────┘
    int i;

}

// IN ANOTHER SOURCE FILE

package p;

class D extends C {

    int f() {return i;}            // can f see i ?   ┌──────────┐
                                                     │ No       │
                                                     └──────────┘

}
```

**Figure 4.3 Active content, first keyword entered.**

Netscape - [Packages]

File   Edit   View   Go   Bookmarks   Options   Directory   Window   Help

2.  same package subclass


// A SOURCE FILE

package p;

| public | class C {

    int i;

}

// IN ANOTHER SOURCE FILE

package p;

class D extends C {

    int f( ) {return i;}          // can f see i ?    | Yes |

}
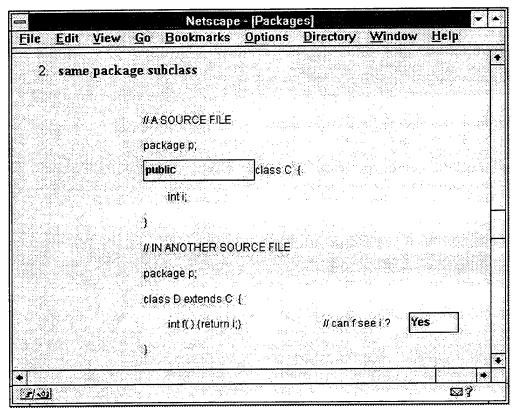
**Figure 4.4  Active content, second keyword entered.**


Netscape - [Packages]

File   Edit   View   Go   Bookmarks   Options   Directory   Window   Help

2.  same package subclass


// A SOURCE FILE

package p;

| public protected | class C {

    int i;

}

// IN ANOTHER SOURCE FILE

package p;

class D extends C {

    int f( ) {return i;}          // can f see i ?    | error |

}
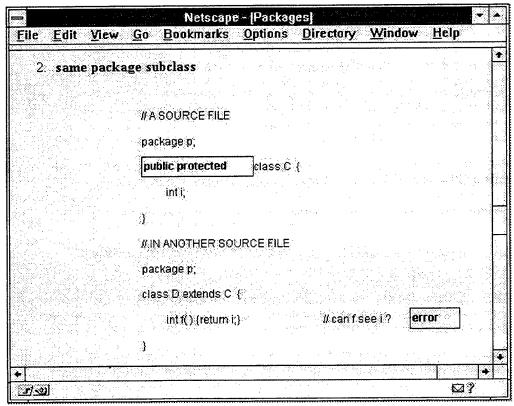
**Figure 4.5  Active content with illegal keyword entered.**

array by the user are selected for use in a calculation. The applet selects different cells based on the dimensions of the array that the user creates.

Figures 4.6-4.8 show one method of doing this with popup windows that are displayed outside the Netscape browser using Java. Notice that the first part of the applet is embedded in the web page like any other. As the demonstration progresses, external windows are created. When the applet is complete, the external windows disappear and the user may start over.
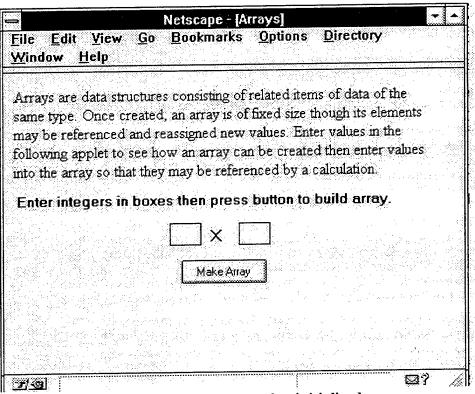


Figure 4.6 Array applet, initialized.

**Netscape - [Arrays]**

**File** **Edit** **View** **Go** **Bookmarks** **Options** **Directory** **Window** **Help**

Arrays are data structures consisting of related items of data of the same type. Once created, an array is of fixed size though its elements may be referenced and reassigned new values. Enter values in the following applet to see how an array can be created then enter values into the array so that they may be referenced by a calculation.

**Enter integers in boxes then press button to build array.**

2 X 4

Make Array

**array window**

**Here is your array.**

2 3 7 6

4 4 3 1

**Enter values for calculation.**

Go!

**Figure 4.7 Array applet, new array created, values entered.**

**array window**

**Here is your array.**

2 3 7 6

4 4 3 1

Enter **calculation results**

From the 2x4 array.

(1,1) + (2,2) =

2 + 4 = 6

Close
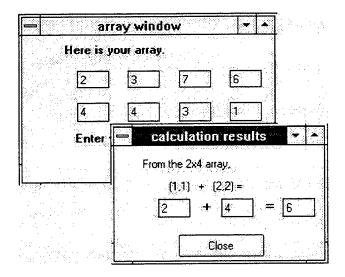
**Figure 4.8 Array applet, elements referenced with result.**

27

## C. SERVER-BASED ACTIVE CONTENT IN COURSEWARE

Previous examples focused on the use of Java applets embedded in Web pages to help explain various concepts. The next step is to afford the user some sort of self assessment. Forms of online examination over networked computers have existed for years. Computerized exams are common in employment centers, computer certification testing centers, and some public schools and can be used anywhere a technical evaluation of knowledge is needed.

One method of accomplishing an enhanced type of online evaluation over the Internet, using a WWW-based system, is to introduce a dedicated course server. We construct a set of lessons, as described in Chapter II. In addition to lesson text and embedded applets, a set of multiple choice questions is prepared to coincide with each topic discussed. An applet is designed to present the question set to the user and process the responses in the browser. A grade is then determined and recorded before proceeding to the next lesson. In order to accomplish this, the question applet communicates with a dedicated course server via a socket connection over the Internet. Upon receiving a connection request, the course server sends a data file to the applet which contains the test questions, the follow-on text URL and the name of the next question file.

In our implementation, the Netscape browser is loaded with a page containing two frames. The upper frame contains the textual content of the lesson. The lower frame contains the question applet which displays a button as shown in Figure 4.9. When pressed, the button launches an external window containing the question set, as shown in Figure 4.10. When the user has completed the question set, the "Grade" button is pressed as shown in Figure 4.11. The questions are scored and the result is shown to the user as in Figure 4.12. When the user clicks "Next Lesson", the URL for the next lesson is passed to the browser and then loaded

into the browser's upper frame. Finally, a data stream, containing the score and the next data

filename, is sent to the course server for parsing.
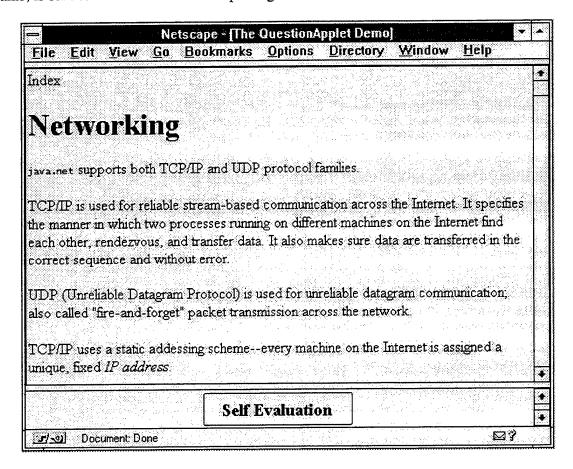


**Figure 4.9 Browser with two frames loaded.**

The course server can be written in any language that supports sockets. In our

implementation, it is written in Java and runs as a background daemon which spawns a new

thread for each connection request from an applet. This allows multiple users to connect

concurrently up to the limitations of the host system. See Figures 3.4 and 3.5 for source code

snippets. Additionally, since this implementation is written in Java, it will work on any

platform that supports the Java Virtual Machine.
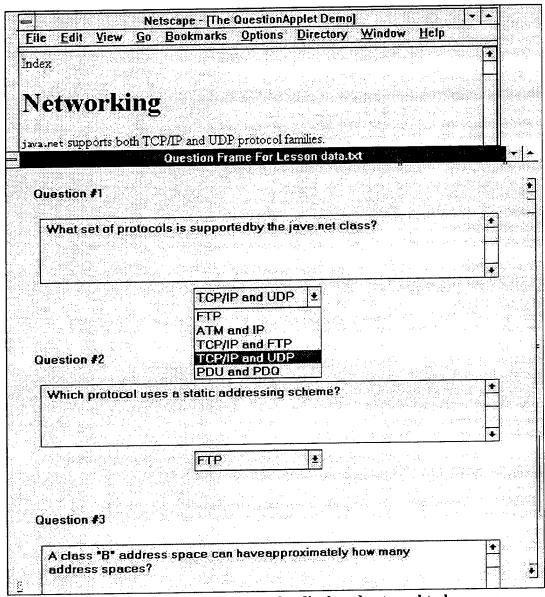
**Figure 4.10 Question applet displayed external to browser**

Since the server is relatively "dumb", it only responds to requests and sends files. It does not maintain any state information about which machine or user is connecting to it. The applet running on the client's machine maintains all state information about the user and subsequent sets of data. Once the question applet classes are initially loaded into the browser, successive question sets load quickly since only the score and the data file pass over the network; the applet itself is not reloaded. In this arrangement, the majority of the workload

takes place on the client machine, which greatly improves course server performance over traditional approaches using HTTP/CGI servers.

Due to current applet security implementations, the course server must reside on the same machine as the applet class files. Thus, the HTTP server and course server are co-located. The data files may reside anywhere that the course server can retrieve them. Once a generic question applet and course server are in place, instructors are only required to place question files and textual materials on the server.
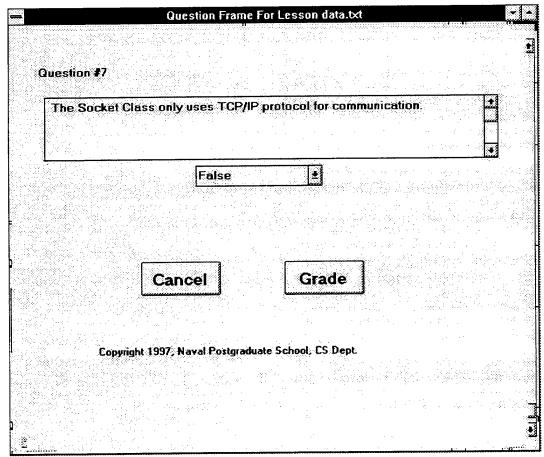


**Figure 4.11 Completion of question set.**
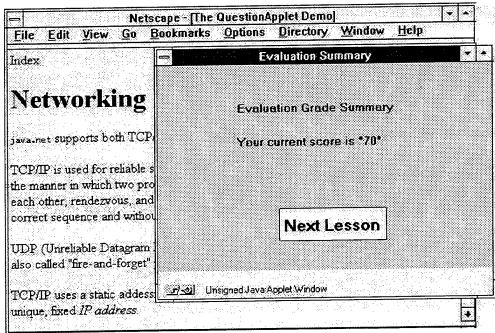
**Figure 4.12 Resulting score of question set.**

# V. FUTURE WORK

The techniques described in Chapter IV can be enhanced in many ways. However, in order to have a fully usable Web-based courseware system, more functionality is needed. We have identified three major areas of improvement. The first two areas involve the QuestionApplet and Course Server. Each should be extended to handle user login authentication. Moreover, the Course Server should provide access to external databases for student identification and course materials. The third area involves facilitating the development of courseware. For this, we propose a tool that we call the Courseware Creation Interface (CCI). Each of these areas is treated in more detail below.

## A.     QUESTION APPLET AND COURSE SERVER

The Question Applet needs to be modified to request user identification and authentication via password, and to pass these parameters to the Course Server. The user identification is used to identify the user to the system. When the user completes a lesson, the score is recorded in a file for review.

## B.     COURSE SERVER USING DATABASES

The Course Server should be modified to uniquely identify student records in a student database. There is already a Java API in place to support connecting any Java application to an existing database using standard database protocols. This functionality would be used, in conjunction with the Question Applet, to validate the user upon login and record lesson scores for review. Additionally, a lesson database should be used to maintain the entire system of lessons and their corresponding evaluation question sets.

Currently, the question sets for each lesson are implemented as sequential ASCII text files with the data members separated by the pipe ( | ) symbol, as shown in Figure 5.1. The Question Applet parses the data file into various standard data elements. The data elements for each lesson, in order, are the number of questions, background color code, foreground color code, follow-on data filename, alternate follow-on filename, next URL, first question, number of possible choices, the multiple choices themselves and the number of the correct choice.

```
4|16777215|0|data.txt|remedial.txt|
http://dubhe.cc.nps.navy.mil/~java/course/net.html|
What type of thread exists to service other threads?|4|red
thread|server thread|daemon thread|green thread|3|
Threads may :|5|yield|sleep|block|pre-empt|all of the above|5|
What is required to make a multiple deposit bank transaction work
properly?|4|serializers|parallel programming|monitor|synchronized
methods|4|
What is happening when a Producer/Consumer is running but nothing
is progressing?|4|deadlock|waiting|livelock|runlock|3|
```

**Figure 5.1  Question data file.**

## C.    A COURSEWARE CREATION INTERFACE

Currently, many instructors fail to implement online courses because of the difficulty in designing and managing the HTML course work. Our idea of an online courseware application abstracts this process to a level that allows anyone to create and maintain online courses.

To this end, we propose a Courseware Creation Interface (CCI). It provides an environment that allows instructors to create sophisticated Web-based courseware. The interface should run within a Java Enhanced browser, such as Netscape or the Microsoft Internet Explorer, and provide a "plug and play" type of format for the various tool components. Given that the courseware is developed in the Java programming language, the CCI will have many advantages, among them, platform-independence.

The CCI should not be designed as a HTML editor, but rather an application used for course standardization, organization, and enhancement. The CCI assumes the user will first produce individual course slides in HTML and then place them in a course package to be exported to the Web or possibly to a CDROM. The course package consists of a course title page, a course index page, the enhanced course pages and other standard courseware components such as a white board, group chat page, bulletin listing, etc. Several screens have been developed as a prototype for reference here, however the "backend" has not been written.

The main screen for the CCI will require the user to click either the "First Time" user button or the "Previous" button (Figure 5.2). By clicking the "First Time" button, the
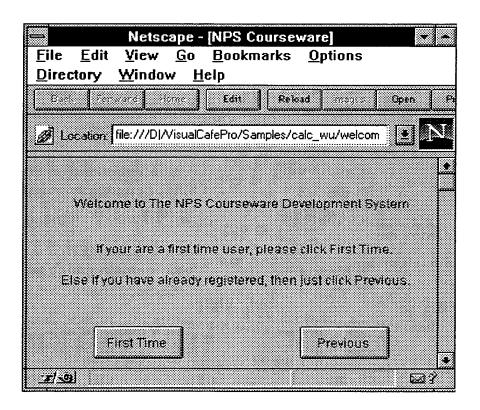


**Figure 5.2 NPS Courseware Initial Screen in Web Page**

35

application displays a registration page requiring the user to input his or her name, user

identification, and a requested password (Figure 5.3).

Once the initial user registration screen is completed, the user clicks on the

"Cancel" or "Register" button. A click of the "Register" button displays the "Login"

frame (Figure 5.4).
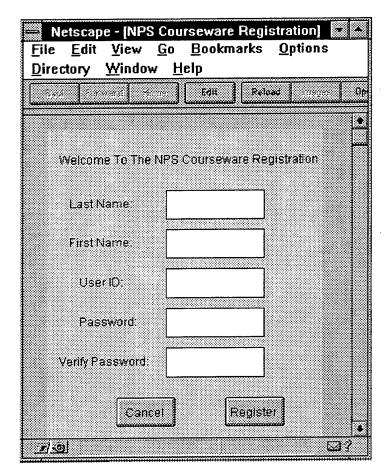


**Figure 5.3 Initial User Registration Screen**

The "Login" screen requires the user to enter his or her user ID and password. If

the user ID or password is incorrect an error frame is displayed. Once the correct user ID

and password are entered and verified, the application displays the course selection frame

(Figure 5.5). After the user selects the course to edit or create, the development frame is

displayed. This frame allows the user to customize the online course by providing course

components for the targeted course (Figure 5.6).

## Login Screen

### NPS Course Development Login Screen

User ID: [_____]

Password: [_____]

[ Login ]    [ Cancel ]
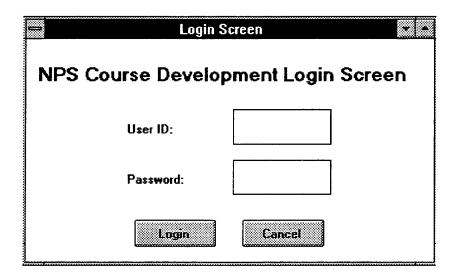
**Figure 5.4  Courseware Login Screen**

## Course Selection Frame

### Course Selection Frame

Select or Enter The Course to Edit or Create

CS2973

[ Continue ]
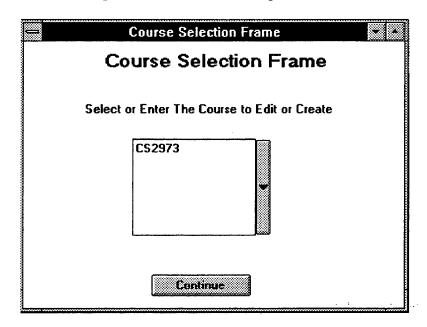
**Figure 5.5  Create New or Edit Existing Course Frame**

For instance, if the user desires to have a whiteboard with the course, the user

simply marks the component with an "X" in the course development frame (see Figure

5.6). The courseware slide organizer tool (Figure 5.7) allows the user to select the slides

**Figure 5.6 Course Development Frame**
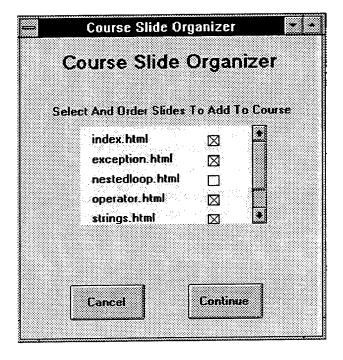


**Figure 5.7  Course Slide Organizer Frame**

(Web pages) that makeup the course by marking the name of the HTML files to include.

Additionally, the user can modify the presentation order of the course slides by

highlighting the desired slide to move, and then "dragging-and-dropping" it into the new

slide ordering. The following provides a more comprehensive definition of the tools

available in the CCI, as shown in Figure 5.6.

1. Navigation Tool

> The navigation tool is used to provide the instructor with the ability to add pre-defined, standardized navigational bars and buttons to the course Web pages. The navigational tools provide hypertext links to the previous page, next page, and course index.

2. Glossary Tool

> The glossary tool allows the instructor to create a searchable glossary of terms. Links from the notes to the glossary entries are added automatically.

3. Course Bulletin Board

> The course bulletin board tool allows the instructor to add a bulletin-board to the course. The bulletin-board allows announcements to be posted, including frequently asked questions and responses. This is useful in any course, but is especially useful when course participants are remotely located.

4. Chat Tool

> The chat tool allows the instructor to implement real-time communication among course participants. This could be used for instructor "office hours" or student communication.

5. Counter Tool

> The counter tool allows the instructor add a page-reference counter to any page of notes and provides some feedback about the amount of usage a page receives. The counter tool has user configurable font size, foreground color, background color, and initial count value.

6. Evaluation Builder Tool

> As mentioned in Sections A and B, the lesson evaluation tool allows the instructor to create lesson evaluations for each lesson. The lesson evaluation tool allows the instructor to implement lesson questions and solutions sets (Figure 5.7). The lesson questions can supplement any page of notes. If there are any questions associated with a page, the courseware generates a button on the bottom of the page. Clicking that button presents the questions in a scrollable frame as described in Chapter IV.

The course author creates questions, potential answers and an explanation for each answer on the Question Creation Frame. Aside from the questions, answers and explanations, the author must indicate which is the correct answer for each question, and with which page this set of questions is associated.

## 7. White Board

The white board tool allows the instructor to create a white board for the course. The white board is a collaborative tool allowing groups of students to create and edit documents, in parallel, over the WWW. Additionally, text and drawing can be created and supported for later re-editing, display, or printing.

## 8. Password Tool

The password tool allows the instructor to create a userid and password for the course or designated pages.

## 9. Help Tool

The help tool allows the user to create a course-specific help index. The help index is active throughout the course and is accessed by clicking the right mouse button.

## 10. Slide Organizer Tool

The slide organizer tool provides the instructor with the ability to organize the slides in a top-down, hierarchical manner. An instructor can select the desired files to be included in the final course build routine and re-position them in the hierarchical structure if needed(Figure 5.7).
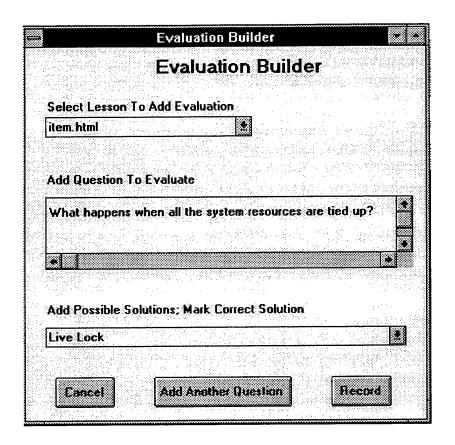
**Figure 5.8 Course Evaluation Creation Frame**

Obviously, the CCI is not a HTML editor, however it does provide limited

functionality in enhancing individual course pages. We propose the following page

courseware tools to insert the appropriate HTML code into CCI files:

1. Electronic Mail

    A electronic mail tool allows the instructor to insert clickable email tags
    into the HTML page. These clickable email tags can be defined to send
    mail to the instructor, individual students or to other predefined email
    addresses.

2. Searchable Image Database

    The searchable Image Database tool provides a user implemented database
    of images that can be linked to course pages. This allows the association
    of names and keywords with each image. Therefore, users can search for
    images based on the name or keywords.

3. Uniform Resource Locator (URL) Tool

    The URL tool provides the ability to place a button-bar icon on any Web
    page linked to an arbitrary URL. This allows arbitrary links to audio,
    images, or other documents.

41

Once a user has completed editing and has clicked the "Course Builder" button,

the course completion frame is displayed (Figure 5.6). The course completion frame

signals that the course was created successfully with the additions of the course title page,

course index page, course selected enhanced pages and courseware component pages

(Figure 5.9).
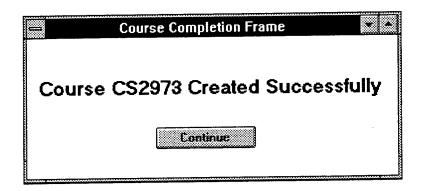


**Figure 5.9  Course Creation Completion Frame**

# VI. RELATED WORK AND CONCLUSIONS

During our studies, we became aware of other Web-based educational courseware implementations. In order to better understand these related works, we classified them into four distinct categories:

1. Administrative Support - web sites that merely display an instructor's administrative material such as class dates, text to read, grading scope, etc. for a regularly scheduled college-level course.

2. Distance Learning Support - course materials that supplement video-based distance learning with Web-based course notes.

3. Tutorial - a self-paced set of Web-based notes that allows a user to independently review and learn from the material at any time.

4. Online Course - a web-based course that allows the user to access the notes of a course and has an active facilitator, usually offered through a university.

The online course is similar to a tutorial with one major exception. In a tutorial, there is no interaction between student and instructor. In contrast, the online course has an instructor that facilitates the course and provides guidance to the students when needed. The following is a list of other Web-based courseware efforts classified according to the categories above.

1. New Jersey Institute of Technology  **http://www.njit.edu/**  (Dist. Learning)
2. The Java Tutorial  **http://pelu.jns.fi/~treijo/javatutor/index.html**(Tutorial)
3. University of California at Berkeley's Chemistry course
   **http://www.cchem.berkeley.edu/~chem130a/**  (Admin)
4. The Globewide Network Academy (GNA)
   **http://uu-gna.mit.edu:8001/uu-gna/index.html** (Admin)
5. Dakota State University - Internet Courses
   **http://www.dsu.edu/distance-ed/internet.html** (Admin)
6. University of  Illinois at Urbana, Classes on the Web
   **http://www.uiuc.edu/webclasses.html** (Online)
7. University of Massachusetts, Dartmouth, CyberEd
   **http://www.umassd.edu/cybered/courses.html** (Online)
8. University of Phoenix Home Page **http://www.uophx.edu/online/** (Online)
9. University of Washington in Seattle

**http://www-hpcc.astro.washington.edu/scied/chemistry.html** (Admin)

10. The Heritage Institute of Seattle, Washington On-line
    **http://www.hol.edu/** (Admin)

11. The Naval Postgraduate School, "Introduction To Object Oriented
    Programming", **http://vislab-www.nps.navy.mil/~java** (Hybrid)

Although the list we have developed is short, given the speed of Web development, we anticipate that this list will continue to grow rapidly.

A key feature of putting all lecture materials in HTML is *real* portability. By using relative URLs within the course material, all pages are initially linked to from an index page. The entire set of pages may be copied to a hard drive and accessed on a stand-alone computer, without a HTTP server or a network and without modifying the code. Furthermore, this same technique could be used to put entire academic or commercial-grade courseware on CDROM.

Although we make several references to CS2973 in this thesis, the course was taught in the classroom, not as courseware. The Web was used as a communication mechanism between the instructor and students, as a medium for visual presentation of the lecture notes in the classroom and as a source of supplemental information for the course material. With the aid of an overhead projector connected to a networked computer display, HTML documents were viewed by students in the classroom through a Web browser. Appendix C describes the resources used in this course. The HTML files remained on the Web server for easy access, future reference, and review by not only the enrolled students, but by anyone from anywhere in the world. CS2973 could be developed into an online course without much additional effort. The notes, assignments and communication mechanisms are already in place.

Departments such as Computer Science have no trouble taking full advantage of the power of the Web for course content delivery. In contrast, other departments less technologically focused, such as National Security Affairs, are either not taking advantage of the Web or are not exploiting its full power. The main difficulty is the initial burden placed upon instructors to completely re-think the nature of their course and to adapt their teaching. The envisioned Java courseware application will offer a tool for educators that provides course standardization, organization, online access, portability and ease of use. By implementing Java courseware that abstracts the complicated details of HTML, Java applets, and Java CGIs, instructors at all levels can take full advantage of the World Wide Web's potential.

Whether or not you agree with the philosophy of online education, the face of higher education is changing. Will our vision of the future come to pass? One thing is certain. The information superhighway is going to have a significant impact on the way universities and organizations educate people. In many ways, education through media, like the Web, enable more people to have access to knowledge than ever before. In this age of information, this is not only a luxury— it is a necessity.

# LIST OF REFERENCES

[AAJava96]     "All About Java", Sun Microsystems,
               **http://java.sun.com/aboutJava/index.html**

[AdvCon94]     Mukesh Singhal & Niranjan G. Shivaratri, "Advanced Concepts In
               Operating Systems", McGraw-Hill Inc., ISBN 0-07-057572

[CoreJava96]   Gary Cornell and Cay S. Horstmann, "Core Java", SunSoft Press, 1996

[Exploring96]  Patrick Niemeyer & Joshua Peck, "Exploring Java",O'Reilly &
               Associates, Inc., 1996

[HandBook96]   Patrick Naughton, "The Java Handbook", McGraw-Hill, Inc., 1996

[HTML/CGI95]   John December & Mark Ginsburg, "HTML & CGI Unleashed", Sams
               Net Publishing, 1995

[JavaPF96]     "The Java Platform, A White Paper", Douglas Kramer,
               **http://java.sun.com/doc/whitePaper.Platform/**
               **JavaPlatform.doc1.html**

[Nutshell96]   David Flanagan, Java in a Nutshell, DavidFlanagan, O'Reilly &
               Associates, Inc., 1996

[NSM95]        "History of the World Wide Web", National Science Museum,
               London, http://www.nmsi.ac.uk/usage/histweb.html 1995

[TM]           Java, JavaO/S, JavaSoft, picoJava, UltraJava, microJava, microJava, Hot
               Java, Sun Microelectronics are Trademarks and Service Marks of Sun
               Microsystems

[Truth81]      H. Broudy, "Truth and Credibility: The Citizen's Dilemma", New York:
               Longman, 1981

## APPENDIX A - COURSE NOTES

The following pages represent the CS2973 Course, "An Introduction To Object Oriented Programming Using Java". These notes were developed jointly by the authors and Professor Dennis Volpano. This notes were created in HTML format and are located at the web page address of http://vislab-www.nps.navy.mil/~java.

# NPS Computer Science Department
# Java Research Index

- CS2973 "Introduction to Object-Oriented Programming using Java"

- Java Cool Applets

- More Java Apps

- API Users Guide

- Post Script Files

- NPS Student Applications/Applets

- Java Course Links

- Java Thesis Research at NPS

- Java Course Director- Dennis Volpano

- Java Course Researchers Rich Moormann and Wes Hester

Email jwhester@cs.nps.navy.mil with questions, comments or suggestions.

Last updated on 21 Apr 96

# CS2973 Course Index

- CS2973 Administrative Notes

- CS2973 Course Syllabus

- CS2973 Homework Assignments

- CS2973 Final Project

- Supplemental Course Information/Examples

- Student Connections

- Helpful Java Links

this page is located at
http://vislab-www.nps.navy.mil/~java/course/index.html

*java@nps.navy.mil*
Last Updated 25 JUL 96
Naval Postgraduate School © 1996

# Introduction to Object-Oriented Programming with Java (CS2973)

# Summer 1996

## Instructor

Dennis Volpano, SP532A
email:volpano@cs.nps.navy.mil

## Teaching Assistants

Wes Hester and Richard Moormann

## Textbooks

The Java Handbook by Patrick Naughton, McGraw Hill, and
Java in a Nutshell by David Flanagan, O'Reilly and Associates Inc. (errata)

We will also be using in the classroom MindQ's CD titled "An Introduction to Programming Java Applets".

## Prerequisites

Every student is required to have an NPS computer account. If you do not have one then get one from the Church Computing Center in Ingersoll Hall. You should have a basic understanding of various UNIX commands needed to manipulate files and directories. It would also help if you were "somewhat" familiar with Web browsers and ubiquitous terms like URL and HTTP. Consider yourself at an advantage if you know how to execute the Netscape Navigator Web browser. Java-deficient browsers like X-Mosaic don't qualify here. As a simple exercise, if you're viewing this with Netscape, click here to visit the CS Java home page, return to this page, and then view the HTML source for this document to see the Java home page URL. What is it?

## Exercises and Final Project

Look Mom. No Exams!

There will be four graded homework exercises. They will constitute 60% of your final grade. The remaining 40% of your grade will be determined by a final project.

Index


this page is located at
http://vislab-www.nps.navy.mil/~java/course/admin.html
*java@nps.navy.mil*
Last Updated 8 JUL96
Naval Postgraduate School © 1996

# CS2973 Syllabus

1. **An Overview of Java**

   O History of the Language

   O The Java Programming Paradigm

2. **The Java Applet**

   O Hello World

   O The Java Developer's Kit (JDK)

   O The APPLET tag and Java-related HTML Syntax

   O The AppletViewer

   O The Scribble Applet (Example 1-2 of Java in a Nutshell)

   O A Simple Text Editor

3. **The Basics of Object-Oriented Programming**

   O What is inheritance?

   O What are classes and objects?

4. **Java Basics** (Chapters 3-6, 9 and 10 of The Java Handbook)

   O Main Method (Chapter 3)

   O Lexical Issues (Chapter 3)

   O Types

   O Simple Types and Arrays (Chapter 4)

   O Strings (Chapter 9)

   O Operators (Chapter 5)

   O Flow Control (Chapter 6)

O Exceptions (Chapter 10)

5. **Java Classes, Packages and Objects** (Chapters 7 and 8)

 O Classes

 O Interfaces

 O Packages

6. **Input/Output** (Chapter 13)

 O File and FileFilter

 O Streams Overview

 O File I/O

7. **Applets** (Chapter 15)

 O Applets or Applications

 O Handling Applet Events (pp. 88-92 of Nutshell)

8. **GUI and the AWT** (Section 5 of Nutshell)
 A recommended AWT article.

 O Components and Containers

 O Subcomponents

 O Panels and Layouts

9. **Java Networking**

 O Sockets

10. **Java Threads**

 O Synchronization and communication

11. **Java Security**

 O The Java security model

Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/syllabus.html

# Object-Oriented Fundamentals

---

- Encapsulation
- Classes and Objects
- Inheritance

The components above are widely believed to capture the essence of object-oriented (OO) programming languages.

We look at each in turn.

## Encapsulation

The idea of hiding representations of data from clients is called **encapsulation.**

Consider the following example.

We need a graphics library for handling polygons, rectangles, circles, etc.

First we introduce polygons:

```
type polygons is some type representing a list of vertices;

procedure translate (p : polygons; a, b : real) is
        // translate every vertex in p by adjusting
        // it horizontally by a and vertically by b
end;

function perimeter (p : polygons) : real is
        // sum lengths between vertices of p
end;
```

Now we use these definitions as in

```
p : polygons;
x, y, r : real;

... translate(p, x, y);

... r := perimeter(p);
```

Here p can be modified directly.

So the program may be *sensitive* to the representation of polygons.

We need to encapsulate the representation. In Ada, this is done as follows:

```
package polygon_package is
        type polygon is private
        procedure translate (...);
        function perimeter(...) returns real;

private
        type polygon is some type;
end polygon_package;

package body polygon_package is
...
end polygon_package;
```

## Classes and Objects

OO languages support encapsulation through classes.

A **class** is like a package specification in Ada with one key difference.

Classes effectively serve as the types of records (tuples) which are *values* in the language.

To use an Ada package, one writes

```
WITH polygon_package;
USE polygon_package;

p : polygon;

... translate(p, x, y);

... r := perimeter(p);
```

In an OO language, the members of a package become the fields of a record:

```
type polygons is record
        v : some type   // vertices field
        procedure translate (a, b : real) is
                // code to adjust vertices in v
        end;
        function perimeter () : real is
```

```
                     // sum lengths between vertices in v
          end;
end record;
```

Notice that `translate` and `perimeter` no longer take a polygon as a parameter:

```
p : polygons;

... p.translate(x, y);

... r := p.perimeter();
```

We must hide the definition of `v` from users of `polygons` so that one cannot write `p.v`.

For this, OO languages furnish quantifiers such as *private* and *public*:

```
type polygons is record
        private v : some type
        public procedure translate (a, b : real) is
                // code to adjust vertices in v
        end;
        public function perimeter () : real is
                // sum lengths between vertices in v
        end;
end record;
```

In OO languages, *record types* are classes.

So `polygons` is a class:

```
class polygons is
        private v : some type
        public procedure translate (a, b : real) is
                // code to adjust vertices in v
        end;
        public function perimeter () : real is
                // sum lengths between vertices in v
        end;
end class;
```

So it is not surprising that Java has no record types.

A record of type (class) `polygons` is called an **instance** or **object** of the class `polygons`.

Objects are "manipulated" at runtime, not classes.

What is the major advantage of encapsulating data via classes?

- Code reuse and flexibility

Suppose we have a specific polygon called `rectangle`:

```
class rectangle is
        private v : some type // 4 vertices
        private side1, side2 : real;
        public procedure create () is
                // create rectangle
        end;
        public procedure translate (a, b : real) is
                // as above
        end;
        public function perimeter () : real is
                return 2 * (side1 + side2);
        end;
end class;
```

Now we would like to be able to write something like

```
p : polygons;

if chosen_icon = rectangle_icon then
        r : rectangle;
        p := r.create;          // a rectangle is a polygon
elseif chosen_icon = circle_icon then
        c : circle;             // a circle is a polygon
        p := c.create;
elseif ...

p.translate(x, y); ... z := p.perimeter();
```

Here the name `perimeter` behaves as though it is **polymorphic**.

It seems to denote a *method* that operates on many different forms of polygons.

In reality, each kind of polygon is bundled with its own `perimeter` method that gets invoked. Which one is invoked by `p.perimeter()` is not known until runtime.

Consider an alternative formulation in a language without objects.

One has to define `polygons` as say a variant record.

The procedure `perimeter` then becomes

```
procedure perimeter (p : polygons) is
```

```
case p.polygontype of

        rectangle : ...
        circle : ...
end;
```

The procedure is *fragile*; adding new `polygontype` requires it to be changed.

# Inheritance

Intuitively, we regard a rectangle as a polygon.

OO programming languages allow us to represent this intuition through **inheritance**.

Inheritance is usually specified by some sort of class extension.

In C++, it is a derived class and in Java it is an extended class.

Reconsider `rectangle`:

```
class rectangle extends polygons
        private side1, side2 : real;
        public procedure create () is
                // as before
        end;
        public function perimeter () : real is
                return 2 * (side1 + side2);
        end;
end class;
```

Now `rectangle` inherits `translate` from `polygons` and **overrides** (shadows) `perimeter` of `polygons`.

Index


this page is located at
http://vislab-www.nps.navy.mil/~java/course/fundamentals.html
*java@nps.navy.mil*
Last Updated 8 JUL96
Naval Postgraduate School © 1996

# Main Method

Sample class with `main()` method:

```
import java.util.Date;
class DateApp {
    public static void main(String args[]) {
        Date today = new Date();
        System.out.println(today);
    }
}
```

A Java application must contain a `main()` method whose signature looks like this

```
public static void main(String args[])
```

The method signature for the `main()` method contains three modifiers:

- `public` indicates that the `main()` method can be called by any object.

- `static` indicates that the `main()` method is a static method (also known as class method).

- `void` indicates that the `main()` method has no return value.

The `main()` method in the Java language is similar to the `main()` function in C and C++. When you execute a C or C++ program, the runtime system starts your program by calling its `main()` function first. The `main()` function then calls all the other functions required to run your program.

Similarly, in the Java language, when you execute a class with the Java interpreter, the runtime system starts by calling the class's `main()` method. The `main()` method then calls all the other methods required to run your application.

If you try to run a class with the Java interpreter that does not have a `main()` method, the interpreter prints an error message.

## Arguments used by the main() Method

The `main()` method accepts a single argument which is an array of Strings.

```
public static void main(String args[])
```

This array of Strings is the mechanism through which the runtime system passes information to your application. Each String in the array is called a **command line argument**. Command line arguments let users affect the operation of the application without recompiling it. For example, a sorting program might allow the user to specify that the data be sorted in descending order with this command line argument:

```
-descending
```

**Note for C and C++ Programmers:** The number and type of arguments passed to the `main()` method in the Java runtime environment differ from the number and type of arguments passed to C and C++'s `main()` function.

Previous I Next I Index I Glossary


this page is located at
http://main.html *java@nps.navy.mil*
Last Updated 15 APR 96
Naval Postgraduate School © 1996

# Strings & StringBuffers

---

What is a **String** or **StringBuffer**?

Think of a **String** or **StringBuffer** as a data structure that stores character data only! However the **String** and **StringBuffer** have some differences.

Java uses two classes that store and manipulate character data.

1. **String** : used for **constant strings**; can't be modified!

2. **StringBuffer**: used for mutable strings; ones that can be modified (reversed, appended too, reduced etc..).

Let's look further at each:

1. String

2. SringBuffer

3. Accessor Methods

The difference is that **Strings** are not mutable wereas **StringBuffers** are mutable!

---

# "String"

---

Derived from the **Class java.lang.String.**

As stated before, the **String** stores a constant **String** value. This means that once you make an assignment to the **String** constant the value can not be modified after creation:

String Name: // creates a variable of type String, Name

The Java compiler insures that each **String** constant actually is a **String** Object. Since **String** Objects are immutable they can be shared:

String Name = "Wes";

creates a variable of type String named Name with an assigned value of Wes which is equivalent to:

```
char data[] = {'W', 'e', 's'};
String temp = new String(data);
```

# "StringBuffer"

Derived from the **Class java.lang.StringBuffer.**

**StringBuffers** are "mutable" - they can be changed, modified, etc! The **StringBuffer** is a growable buffer for characters. It is mainly used to create **Strings**. The Java compiler uses it to implement the "+" operator.

For example:

"W" + 4 + "s";

is compiled to:

new StringBuffer().append("W").append(4).append("c").toString()

Note that the method toString() does not create a copy of the internal buffer. Instead the buffer is marked as shared. Any further changes to the buffer will cause a copy to be made.

Declaring **StringBuffers**

```
int len = 5;
StingBuffer name = new StringBuffer(length_to_make_StringBuffer);
```

```
Now we have just created an object called name of type StringBuffer. Additionally,
StringBuffer objects can be created with the java.lang.StringBuffer Constructors.
Sample "StringBuffer" file
```

# "Accessor Methods"

**Accessor Methods** are the methods used to obtain information about an object. For example:

```
public static String traverseIt(String source)
    {

    int i, len = source.length();

    StringBuffer dest = new StringBuffer(len);

    for (i = 0; i < len; i++)

    {

        dest.append(source.charAt(i));

    }

    return dest.toString();

    }
```

Above there are two **Accessor Methods**:
   1. int len = source.length();

   2. dest.append(source.charAt(i));

An object's instance variables are encapsulated within the object, hidden inside, safe from inspection or manipulation by other objects. With certain well-defined exceptions, the object's methods are the only means by which other objects can inspect or alter an object's instance variables. Encapsulation of an object's data protects the object from corruption by other objects and conceals an object's implementation details from outsiders. This encapsulation of data behind an object's methods is one of the cornerstones of object-oriented programming.

Sample **"Accessor Method"** file

<div align="center">Top</div>

---

# Operators

---

In the world of automation there are three (3) types of operation notations:

1. Infix

2. Prefix

3. Postfix

However, Java uses **infix** notation to perform a given operation on two operands. Example: Java **Operators** are classified into four (4) categories:

1. **Arithmetic Operators**: [ *,+,-,/,%,^] The Java language supports various arithmetic operations--including addition, subtraction, multiplication, and division--on all numbers. The statement count++ uses a short cut operator ++, which increments a number.

2. **Relational Operators**: [ ==, !=, <, >, <=, >=] The relational operators compare two values and determine the relationship between them. For example, != returns true if two values are unequal. The character-counting program uses != to determine whether the value returned by System.in.read() is not equal to -1.

3. **Logical Operators**: [ &&, ||] The logical operators take two values and perform boolean logic operations. Two such operators are && and ||, which perform boolean and and boolean or operations, respectively. Typically, programmers use logical operators to evaluate compound expressions. For example, this code snippet verifies that an array index is between two boundaries:

        if (0 < index && index < NUM_ENTRIES)

4. **String Operators**: [ + ] The Java language extends the definition of the operator + to include string concatenation. The example program uses + to contenate "Input has ", the value of count, and " chars."

        System.out.println("Input has " + count + " chars.");

**Operator Precedence**: The Java language allows you to create compound expressions and statements such as this one:

        x * y * z

In this particular example, the order in which the expression is evaluated is unimportant because multiplication is commutative. However, this is not true of all expressions, for example:

        x * y / 100

gives different results if you perform the multiplication first or the division first. You can use balanced parentheses ( and ) to explicitly tell the Java compiler the order in which to evaluate an expression, for example x * (y / 100), or you can rely on the precedence the Java language assigns to each operator. This chart illustrates the relative precedence for all Java operators.

# Precedence Chart

(in decreasing order of precedence)

| Precedence | Operator | Description | Type | Associativity |
|---|---|---|---|---|
| 1 | ++ | Pre-or-post increment | Arithmetic | Right to Left |
| 1 | -- | Pre-or-post decrement | Arithmetic | Right to Left |
| 1 | +,- | unary plus, minus | Arithmetic | Right to Left |
| 1 | ~ | Bitwise Complement (unary) | Integral | Right to Left |
| 1 | ! | Logical Complement (unary) | Boolean | Right to Left |
| 1 | (type) | Cast to a type | Any Type | Right to Left |
| 2 | *, /, % | Multiplication, Division, Remainder (Mod) | Arithmetic | Left to Right |
| 3 | +, - | Addition, Subtraction | Arithmetic | Left to Right |
| 3 | + | String Concatenation | String | Left to Right |
| 4 | << | Left Shift | Integral | Left to Right |
| 4 | >> | Right Shift with sign extension | Integral | Left to Right |
| 4 | >>> | Right Shift with zero extension | Integral | Left to Right |
| 5 | <, <= | Less than, Less than equal | Arithmetic | Left to Right |

| | | | | |
|---|---|---|---|---|
| 5 | >, >= | Greater than, Greater than equal | Arithmetic | Left to Right |
| 5 | instanceof | Type of comparison | Object Type | Left to Right |
| 6 | == | Equal to | Primitive | Left to Right |
| 6 | == | Equal to | Object | Left to Right |
| 6 | != | Not equal to | Object | Left to Right |
| 7 | & | Bitwise AND | Integral | Left to Right |
| 7 | & | Boolean AND | Boolean | Left to Right |
| 8 | ^ | Bitwise XOR | Integral | Left to Right |
| 8 | ^ | Boolean XOR | Boolean | Left to Right |
| 9 | I | Bitwise OR | Integral | Left to Right |
| 9 | I· | Boolean OR | Boolean | Left to Right |
| 10 | && | Conditional AND | Boolean | Left to Right |
| 11 | II | Conditional OR | Boolean | Left to Right |
| 12 | ?: | Conditional (ternary) operator | Boolean, any | Right to Left |
| 13 | =, *=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, I= | Assignment with operation | Variable, any | Right ot Left |
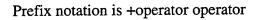
Top

# Infix Notation

Infix notation is operator + operator

ie. 10 = 5 + 5

# Prefix Notation

Prefix notation is +operator operator

ie. 10 = + 5 5

---

Top

# Postfix Notation

Postfix notation is **operator operator +**

ie. 10 = 5 5 +

---

Top

---

Index

# Exceptions

---

## I. What is an Exception?

(Java Handbook, Chap 10, pg 165)

An event that occurs during the execution of the program that prevents the continuation of the normal flow of instructions, such as the divide by 0 (ie 4/0).

Different computer systems handle exceptions in different ways; some more elegantly than others.

Exceptions are used for changing the flow of the control when some important or unexecpted event, usaully an error, occurs.

Exceptions devert processing to a part of the program that can try to handle the error. *(sort of like a goto statement....)*

The **"exception"** Control Flow Statement is used to handle an error or warn potential callers of the method about the possiblity of an error via a **throws** statement.

In past programming languages the programmer would try to solve the exception problem by creating an error check for each input:

```
int status = something_that_you_know_works();

if (status == Bad_return_value)
    {
        switch(some_global_exception_indicator)
        {
            handle_excepton_someway;
        }
    }
else
    {
            all_is_well(continue_happily);
    }
```

Top

---

## II Fundamentals

Exceptions are caused in one of two ways:

1. Program does something illegal (usual case)..Implicit Exception:
2. Program explicitly generates an exception by executing the **throw** statement (less usual case).

Here is the basic form of an exception handling block:

```
try {
    // block of code
}

catch ( ExceptionType1 e) {
    // exception handler for ExceptionType1
}

catch ( ExceptionType2 e) {
    // exception handler for ExceptionType2
    throw(e); // re-throw the exception
}

finally {
    // always done
}
```

Top

---

## III. Exception Types

There is a single class at the top of the exception class hierarchy called Throwable This class is used to represent all exceptional conditions, with each ExceptionType in general a subclass of Throwable.

There are a number of predefined exceptions in the Java language:

Class `Throwable` encompasses objects that may be thrown by the `throw` statement.
`java.lang.Throwable` is the root class of the Java exception and error hierarchy. All exceptions and errors that are thrown are subclasses of **Throwable.**

The hierarchy of classes defined in package `java.lang` are as follows.

**java.lang.Throwable**

```
        Error
            LinkageError
                    ClassCircularityError
                    ClassFormatError
                    ExceptionInInitializerError
                    IncompatibleClassChangeError
                            AbstractMethodError
                            IllegalAccessError
                            InstantiationError
                            NoSuchFieldError
                            NoSuchMethodError
                    NoClassDefFoundError
                    UnsatisfiedLinkError
```

```
                    VerifyError
              VirtualMachineError
                    InternalError
                    OutOfMemoryError
                    StackOverflowError
                    UnknownError
                ThreadDeath
          Exception
                ClassNotFoundException
                CloneNotSupportedException
                IllegalAccessException
                InstantiationException
                InterruptedException
                NoSuchMethodException
                RuntimeException
                        ArithmeticException
                ArrayStoreException
                        ClassCastException
                        IllegalArgumentException
                                NumberFormatException
                                IllegalThreadStateException
                        IndexOutOfBoundsException
                        NegativeArraySizeException
                        NullPointerException
                        SecurityException
```

Top

# IV. Uncaught Exceptions

Exception objects are automatically created by the Java runtime in response to some exceptional condition. For example:

```
class Exc0 {
   public static void main(String args[]) {
      int d = 0;
      int a = 42/d; // exception, divide by zero
   }
}
```

The runtime tries to execute the division, causing a new exception object to throw a

We say throws as a "hot potato", that the code must catch and deal with immediately.

An exception handler would handle this exception, but in this example we
don't have an exception handler, thus it defaults to the runtime handler.
The default runtime handler then prints out the following:

```
  c:> java Exc0
  java.lang.ArithmeticException: / by zero
          at Exc0.main(Exc0.java:4)
```

Note: The name Exc0, method name main, and the line number 4 that the error was foun

Also the Exception that was thrown was called ArithmeticException, which is a  subcl
of Throwable.


Another example is the same class which introduces the same error but in
a separate method:


```
class Excl {
    static void subroutine() {
        int d = 0;
        int a = 10 / d;
    }
    public static void main(String args[]) {
        Excl.subroutine();
    }
}
```

The call stack trace from running Excl.java is:

```
c:> java Excl
java.lang.ArithmeticException: / by zero
        at Excl.subroutine(Excl.java:4)
        at Excl.main(Excl.java:7)
```

Note that the bottom of the stack is in the main method, line 7, which is called
form line 4 of the subroutine() method.

<center>Top</center>

---

# V. try and catch

The Java runtime handles the exception, however it is more desirable to handle the
exception yourself.

Four (4) key statements associated with **exceptions**:

- try - runs the code inside the braces, if there is an error then use the catch
  to tell me how to handle them. The try statement governs the statements
  enclosed within it and defines the scope of any **exception** handlers (established
  by subsequent catch) associated with it.

- catch - allows you to handle any and all exceptions that are instances. You
  associate exception handlers with a try block by providing one or more
  sequential catch blocks directly after the try block.

- throws - sort of like a break statement, nothing beyond it is executed.

- finally - used to perform some action that you absolutely have to do (ie. to
  free some external resource) To be sure you use the finally statement. Java's
  finally statement provides a mechanism which allows your method to cleanup
  after itself regardless of what happens within the try block. Use the finally
  statement to close files or release other system resources.

  For now lets look at the try and catch statements.

          try

```
        {
         do_something();
        }
        catch (SomeThrowableClassName variableNames)
        {
            do_something;
        }
        catch (SomeThrowableClassName variableNames)
        {
            do_something;
        }
```

Exc2.java example of try and catch

The scope of the catch clause is restricted to those statements specified by
the immediately proceding try statement.

Also, it is advised to use { (begin blocks) and } (end blocks) with try and
catch to make the code more readable.

<div align="center">Top</div>

---

# VI. Multiple catch Clauses

In some cases you may desire to have more than one exception condition raised.

You can have any number of catch clauses in a row. The following example traps
two different exception types followed by an all-purpose handler for all
Throwable types: (page 170)

MultiCatch.java same code

Note: This example catches the divide by zero and array out of bounds
exceptions.

<div align="center">Top</div>

---

# VII. Nested try Statements
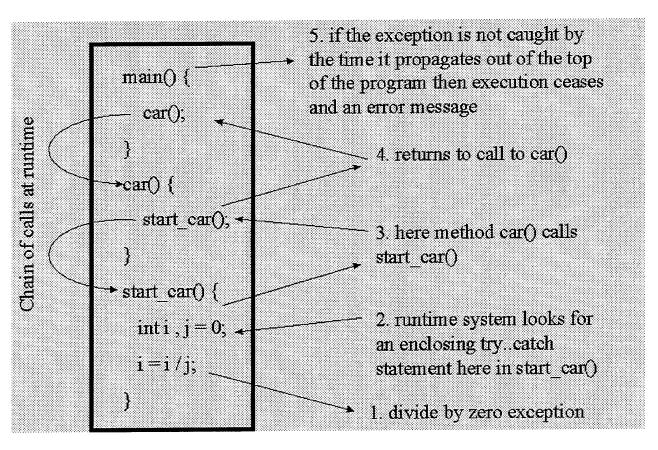
The try statement nests similar to variable scope.

You can wrap a try statement around a method call, and inside that method,
someother try statement might protect some other code.

Each time a try statement is entered, the context of that exception frame is
stacked up until all of the nexted try statements complete.

If a lower level try statement does not have a catch handler for a particular
exception, the stack is unwound and the next try statements catch are inspected
for a match. This continues until there is a match, else it hits the main
method and a runtime error is generated.

Here is a graphic example:

Chain of calls at runtime

```
main() {
    car();
}

car() {
    start_car();
}

start_car() {
    int i , j = 0;
    i = i / j;
}
```

5. if the exception is not caught by the time it propagates out of the top of the program then execution ceases and an error message

4. returns to call to car()

3. here method car() calls start_car()

2. runtime system looks for an enclosing try..catch statement here in start_car()

1. divide by zero exception

MultiNest.java, page 170

Top

---

## VIII. throw

The throw statement is used to explicitly throw an exception.

The general form is :

```
        throws ExceptionObject;
              |
              |
              V
        throws java.io.IOException;
```

The ExceptionObject (IOException) is an object of a class that extends the class **java.lang.Exception.**

The flow of the execution stops immediately after the throw statement, and the next statement is not reached.

The closest try block is inspected to see if it has a catch clause which matches the type of the Throwable instance. If it does, then control is transferred to that statement, else the next enclosing try statement is inspected and so on..until the outermost exception handler halts the program.

Here is an example of throwing a new instance of an exception:

ThrowDemo.java, page 171

---

## IX. throws

If a method is capable of causing an exception which it does not itself handle, then it should notify the methods above of the exception so that callers can guard themselves against this exception.

The throws keyword is used to identify the list of possible exceptions that might be thrown by a method.

Here is an example of a program that tries to throw an exception without having any code to catch it, nor using throws to declare that the exception will be thrown..this will not compile:

```
class ThrowsDemo1 {
  static void procedure () {
    System.out.println("inside procedure");
    throw new IllegalAccessException("demo");   //trys to throw without catch
  }

  public static void main (String args[]) {
    procedure();
  }

}
```

Inorder to make this compile we need to declare that procedure throws IllegalAccessException and add a try and catch statement.

For example:

ThrowsDemo.java, pg 173

---

## X. finally

When exceptions are thrown, the flow of code in a method takes a non-linear flow through the method, skipping lines.

To assure that a given piece of code will run no matter what, the finally keyword can be used to identify a block of code to always be executed..ie closing a file that is open.

Even if there is no catch clause, the finally statement will be executed before the code block ends.

The finally statement is not manditory, but optional.

Here is an example:

FinallyDemo.java, pg 174

---

# XI. Exception Subclasses

Java allows the use of User-Defined Exceptions created to uniquely handle possible errors in execution.

Only subclasses of class Throwable may be caught or thrown. Simple types, such int or char, as well as non-Throwable classes such as String and Object may not be used as exceptions.

For example:

```
    class OutofGas extends Exception {
                        System.out.println("Exception here");
    }

    class car {

        // do something

        if (fuellevel < 0.5) throw new OutofGas();   // throw exception

    }
```

Any method that throws a User-Defined exception must also catch it. The advantage of using exceptions is that you can collect all errors in well-localized places in your program...however, it does not always reduce the work.

Another example:

ExceptionDemo.java, pg 175

---

# XII. Summary of Exceptions

O Purpose: Safer programming by providing a distinct path to deal with errors
O Do Use Them: They provide a useful tool for organizing error handling.
O Exceptions' main use is to get a decent error message out explaining what failed and where and why. Exceptions don't take all the work out of finding errors, but they do allow you to put errors in neat packages...thus handling the errors and exceptions.

---

this page is located at
http://vislab-www.nps.navy.mil/~java/course/exceptions.html

# Classes

A Java program is a collection of *class* definitions.

A class definition has the form

```
class classname [extends superclassname] {
        type var1;
        type var2;
        type varN;
        type methodname1(parameter-list) {
                method-body;
        }
        type methodname2(parameter-list) {
                method-body;
        }
        type methodnameN(parameter-list) {
                method-body;
        }
}
```

Unlike C++, class declarations and method implementations are not separated.

Instances of classes are created at run time and are called **objects**.

## Instance Variables

Variables var1, var2,..., varN are called **instance variables**.

Every instance of the class has its own set of these variables.

```
class Point {
        int x, y;
}
```

Every instance of Point has a variable called x and one called y.

How do we create instances of a class? By the **new** operator.

The **new** operator creates a single instance of a class and returns a *reference* to that object. The instance variables of the object are accessed using dot notation.

```
Point p = new Point( ); // creates a new instance of Point; reference stored in p

Point p2 = p;           // p and p2 now store a reference to the same object

p2.x = 17;              // p.x == p2.x == 17

Point p3 = new Point( );
p3.x = 5;               // p2.x == 17   && p3.x == 5
p = null;               // p2.x == 17
```

```
p.x = p2.x;              // throws NullPointerException
```

Two instances of Point are created in class TwoPoints.java.

## Object cloning

We can redefine Point to implement class **Cloneable** so that **copies** of Point objects are possible:

```
class Point implements Cloneable {
        int x, y;
        public Object clone( ) {
                Point p = new Point( );
                p.x = this.x;
                p.y = this.y;
                return p;
        }
}
class CloneDemo {
        public static void main(String args [ ]) {
                Point p1 = new Point( );
                p1.x = 16;

                Point p2 = (Point) p1.clone( );
                p1.x = 32;
                System.out.println("p2.x = " + p2.x);    // outputs 16
        }
}
```

## Java records

Java has no traditional records (structures). Instead one uses classes with instance variables.

```
class LinkedList {
        int x;
        LinkedList next;
}
class ListDemo {
        public static void main(String args[ ]) {
                LinkedList myList = new LinkedList( );
                myList.x = 16;
                myList.next = new LinkedList( );
                myList.next.x = 32;
                myList.next.next = null;

                for(LinkedList p = myList; p != null; p = p.next)
                        System.out.println(p.x);             // outputs 16, 32
        }
}
```

## Instance Method Declaration and Access

In addition to instance variables, classes may contain **instance methods**.

```
class Point {
        int x, y;
        void init(int a, int b) {        // an instance method
                x = a;
```

```
                    y = b;
        }
}
class PointDemo {
        public static void main(String args[ ]) {
                Point p = new Point( );
                p.init(10, 20); // call p's init instance method
        }
}
```

As another example, consider a circle class with two instance methods:

```
class Circle {
        double x, y;     // center
        double r;        // radius
        // two instance methods
        double circumference( ) { return 2 * 3.14159 * r;}
        double area( ) {return 3.14159 * r * r;}
}
class CircleDemo {
        public static void main(String args [ ]) {
                Circle c = new Circle( );
                double a;
                c.x = 2.0;
                c.y = 2.0;
                c.r = 1.0;
                a = c.area( );   // access c's instance method area( )
        }
}
```

## Instance Methods and "this"

An instance method is implemented with an implicit argument called **this**:

```
double circumference(Circle this) { return 2 * 3.14159 * this.r;}
double area(Circle this) {return 3.14159 * this.r * this.r;}
Circle c = new Circle( );
a = area(c);
```

**this** is not explicit in instance method signatures because it is assumed that when a method accesses any instance variable or method in its class, it is accessing the fields in the object referred to by the **this** argument.

In some cases, **this** is required for scope resolution:

```
class Point {
        int x, y;
        void init(int x, int y) {
                this.x = x;
                this.y = y;
        }
}
```

## Constructors

A **constructor** is a special method for initializing an object immediately upon creation. It has the same

name as the class in which it appears and has NO RETURN TYPE.

A constructor's job is to initialize all internal state.

## Three examples of constructors

1. 
```
class Point {
        int x, y;
        Point(int x, int y) {    // a constructor method
                this.x = x;
                this.y = y;
        }
}
class PointDemo {
        public static void main(String args[ ]) {
                Point p = new Point(10, 20);
        }
}
```

2. 
```
class Circle {
        double x, y, r;
        Circle(double x, double y, double r) { // a constructor method
                this.x = x;
                this.y = y;
                this.r = r;
        }
        // two instance methods
        double circumference( ) { return 2 * 3.14159 * r;}
        double area( ) {return 3.14159 * r * r;}
}
class CircleDemo {
        public static void main(String args [ ]) {
                Circle c = new Circle(2.0, 2.0, 1.0);
                double a;
                a = c.area( );
        }
}
```

3. 
```
class LinkedList {
        int x;
        LinkedList next;
        LinkedList(int x, LinkedList p) {        // a constructor method
                this.x = x;
                this.next = p;
        }
}
class ListDemo {
        public static void main(String args[ ]) {
                // LinkedList myList = new LinkedList( );
                // myList.x = 16;
                // myList.next = new LinkedList( );
                // myList.next.x = 32;
                // myList.next.next = null;

                LinkedList myList = new LinkedList(16, new LinkedList(32, null)

                for(LinkedList p = myList; p != null; p = p.next)
                        System.out.println(p.x);
        }
```

```
      }
```

## Destructors

The Java-equivalent notion of the C++ destructor is the instance method **finalize**.

It is executed just before an object of the class is garbage collected.

Used to free up system resources like I/O streams (file descriptors and sockets), but not memory since it is reclaimed by the runtime system.

Here is the finalizer method from FileOutputStream:

```
protected void finalize( ) throws IOException {
        if (fd != null) close( );
}
```

## The Default Constructor

If a constructor is not defined for a class, then the default constructor is automatically inserted which merely calls the superclass constructor:

```
Point( ) {super( );}     // call to empty constructor Object( )
Circle( ) {super( );}
```

# Method Overloading

The same name can be used for more than one method -- called **method overloading**.

Overloaded names are resolved through a method's type signature, as in Ada.

A type signature comprises a method's arity and parameter types.

The name "Point" is overloaded in the constructor methods of PointCreateAlt.java.

The second definition of Point:

```
        Point( ) {x = y = -1;}
```

can be rewritten using **this** as

```
        Point( ) {this(-1, -1);}        // call to first Point constructor
```

Use of **this** within a constructor is RESTRICTED to appear as the first statement.

The reason is to ensure that all superclass constructors are executed first so that the state they control is initialized.

Method overloading is not limited to constructors.

See overloading of "distance" method in PointDist.java.

# Class Extension

Class extension allows one class to **inherit** the instance variables and methods of another class (single inheritance).

A class' immediate parent is called its **superclass**.

```
class Point {
  int x;
  int y;
  public boolean equal(Point p) {
    return (x==p.x && y==p.y);
  }
}
class ColPoint extends Point {
  int color;
  // overload equal
  public boolean equal(ColPoint p) {
    return (super.equal(p) && color==p.color);   // x==p.x && y==p.y && color==p.colo
  }
}
public class ColoredPoints {
  public static void main(String[] args) {
    ColPoint cp = new ColPoint( );
    Point p = new Point( );

    System.out.println(cp.equal(p));
  }
}
```

Contrast the use of **super** above with its use as a constructor in Point3D.java.

# Method Overriding

When a class defines a method using the same name, return type, and arguments as a method in its superclass, the method in the class **overrides** the method in the superclass.

The new method is called for objects of the class, not the superclass' old definition.

```
class A {
        int i = 1;
        int f( ) {return i;}
}
class B extends A {
        int i = 2;                      // shadows variable i in class A
        int f( ) {                      // overrides method f in class A
                i = super.i + 2;
                return super.f( ) + i;
        }
}
class OverRideDemo {
        public static void main(String args[ ]) {
                B b = new B( );
                System.out.println(b.i);          // outputs 2
                System.out.println(((A)b).i);     // outputs 1
```

```
                System.out.println(b.f( ));        // outputs 4
        }
}
```

Now take a look at the example in Point3DDist.java.

The preceding example can be explained by making **this** explicit:

```
class Point {
        int x, y;
        Point(Point this, int x, int y) {
                this.x = x;
                this.y = y;
        }
        double distance(Point this, int x, int y) {
                int dx = this.x - x;
                int dy = this.y - y;
                return Math.sqrt(dx * dx + dy * dy);
        }
        double distance(Point this, Point p) {
                return distance(this, p.x, p.y);            // dynamic method dispatch h
        }
class Point3D extends Point {
        int z;
        Point3D(Point3D this, int x, int y, int z) {
                super(this, x, y);
                this.z = z;
        }
        double distance(Point3D this, int x, int y, int z) {
                int dx = this.x - x;
                int dy = this.y - y;
                int dz = this.z - z;
                return Math.sqrt(dx * dx + dy * dy + dz * dz);
        }
        double distance(Point3D this, Point3D other) {
                return distance(this, other.x, other.y, other.z);
        }
        double distance(Point3D this, int x, int y) {
                double dx = (this.x / z) - x;
                double dy = (this.y / z) - y;
                return Math.sqrt(dx * dx + dy * dy);
        }
class Point3DDist {
        public static void main(String args [ ]) {
                Point3D p1 = new Point3D(30, 40, 10);
                Point p = new Point(4, 6);
                . . . distance(p1, p) . . .
        }
}
```

**Dynamic method dispatch** takes place among **instance** methods that are defined with the same name in **subclasses** of a given class.

Above, we have Point3D is a subclass of Point, so distance(p1, p) results in a call to the first distance method of Point.

The call distance(this, p.x, p.y) is not resolved **statically** as one might expect with **this** having type Point.

Instead the call is resolved dynamically by retaining at runtime, the type of the object whose reference is stored in **this**, namely Point3D.

Notice that technically speaking, 2D distance is not really overridden by the definition given in Nutshell (pg. 68) since the first distance definition in Point and the third given in Point3D actually have different type signatures when **this** is made explicit!

As another example of dynamic method dispatch, see Dispatch.java.

### Final instance methods and variables

All instance variables and methods can be overridden by default.

To prevent one of these from being overridden, use the **final** qualifier.

The following is taken from class **ThreadGroup**:

```
public final String getName( );
public final ThreadGroup getParent( );
public final synchronized void stop( );
```

**final** is also used to prevent a class from being extended:

```
public final class String extends Object {. . .}
```

Using **final** in these ways is aimed at preventing Java programs from extending Java's built-in methods in malicious ways that could compromise security.

It may also be used for a more traditional purpose: to get constants:

```
final int FILE_NEW = 1;          // final instance variables
final int FILE_QUIT = 5;
```

Finally, all **static** and **private** methods are implicitly final.

# Class Variables and Methods

So far we have considered only instance variables and instance methods.

Each object has its own instance of an instance variable and its own instance of an instance method.

There is another kind of variable called a **class variable** and another kind of method called a **class method**. Each is indicated as such in Java using the keyword **static**.

### Class variables

A class variable is shared among all instances of a class. Only one copy of the variable exists regardless of the number of instances of the class that are created.

```
public class Circle {
        static int num_circles = 0;       // counter class variable
        public static final double PI = 3.14159265;      // final class variable
        public double x, y, r;

        public Circle(double x, double y, double r) {
                this.x = x; this.y = y; this.r = r;
                num_circles++;  // no need to say Circle.num_circles within class
        }
}
public class CircleDemo {
        public static void main(String args [ ]) {
                System.out.println(Circle.num_circles); // no instance of Circle nee
                double circum = 2 * Circle.PI * radius; // or here
        }
}
```

## Class methods

Class methods differ from instance methods in that they ARE NOT PASSED AN IMPLICIT **this** reference.

These methods are not associated with any instance of the class. There is only ONE copy for all instances.

This means that class methods

1. cannot call instance methods
2. cannot access instance variables
3. cannot refer to **this** or **super**
4. can have their calls optimized by inlining method byte codes

We have already seen an example of a class method: Math.sqrt( ). In fact, EVERY method in class Math is a class method (see pg. 317 of Nutshell).

As another example, consider

```
public final class System extends Object {
        public static PrintStream out;
        . . .
}
public class PrintStream extends FilterOutputStream {
        public synchronized void println(char c);
        public synchronized void println(String s);
        . . .
}
```

## Abstract methods

Java lets you **specify** a method without defining it by making it **abstract**.

Abstract methods are used when you wish to specify the operations of a class but do not have enough information to actually define them.

```
public abstract class Shape {
      . public abstract double area( );
        public abstract double circumference( );
}
public class Circle extends Shape {
        double r;
        static final double PI = 3.14159265;
        public double area( ) {return PI * r * r;}
        public double circumference( ) {return 2 * PI * r;}
}
public class Rectangle extends Shape {
        double w, h;
        public double area( ) {return w * h;}
        public double circumference( ) {return 2 * (w + h);}
}
```

Some facts to remember about abstract methods:

1. Any class with an abstract method is **abstract** itself.
2. An abstract class may define some non-abstract methods.
3. An abstract class cannot be instantiated.
4. A subclass of an abstract method can be instantiated if it defines every abstract method of its superclass.

See the class methods in Static.java and StaticByName.java.

Index


this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw.html

*java@nps.navy.mil*
Last Updated 25 JUL96
Naval Postgraduate School © 1996

# Interfaces

Class extension in Java provides single inheritance only.

That is, a class can inherit method implementations from exactly one superclass.

Multiple inheritance allows for a class to inherit method implementations from one or more superclasses, as in C++.

The Java designers feel that inheriting multiple method **implementations** is not all that useful.

However, the Java designers do feel that inheriting multiple method **signatures** is useful.

Think of a method signature as the method's name, parameters, and return type.

To support this kind of inheritance, Java uses **interfaces**.

An **interface** is an abstract class.

An abstract class may define some non-abstract methods, however, **all** the methods in an interface are implicitly abstract.

## Example

Recall the abstract classes:

```
public abstract class Shape {
        public abstract double area( );
        public abstract double circumference( );
}
public class Circle extends Shape {
        protected double r;
        Circle( ) {r = 1.0;}
        Circle(double r) {this.r = r;}
        static final double PI = 3.14159265;
        public double area( ) {return PI * r * r;}
        public double circumference( ) {return 2 * PI * r;}
}
public class Rectangle extends Shape {
        protected double w, h;
        Rectangle( ) {this(0.0, 0.0);}
        Rectangle(double w, double h) {
                this.w = w;
                this.h = h;
        }
        public double area( ) {return w * h;}
        public double circumference( ) {return 2 * (w + h);}
}
```

And we can use them as follows:

```
Shape[ ] shapes = new Shape[3];
shapes[0] = new Circle(2.0);
shapes[1] = new Rectangle(1.0, 3.0);
shapes[2] = new Rectangle(4.0, 2.0);

double total_area = 0;
for (int i = 0; i < shapes.length; i++)
        total_area += shapes[i].area( );          // dynamic dispatch
```

Now suppose we wish to extend our package of shapes by allowing them to be drawn. So we do the following:

```
public abstract class DrawableShape {
        public abstract void setColor(Color c);
        public abstract void setPosition(double x, double y);
        public abstract void draw(DrawWindow dw);        //DrawWindow defined elsewhe
}
public class DrawableRectangle extends DrawableShape {
        private protected double w, h;
        private protected Color c;
        private protected double x, y;   // position
        DrawableRectangle(double w, double h) {
                this.w = w;
                this.h = h;
        }
        public double area( ) {return w * h;}
        public double circumference( ) {return 2 * (w + h);}

        // define abstract methods of DrawableShape
        public void setColor(Color c) {this.c = c;}
        public void setPosition(double x, double y) {
                this.x = x;
                this.y = y;
        }
        public void draw(DrawWindow dw) {
                dw.drawRect(x, y, w, h, c);
        }
}
```

But we don't want to duplicate Rectangle's variables and methods, namely w, h, area( ) and circumference( ), as above.

Instead, we should inherit them from Rectangle.

But we can't since that would require we write

```
public class DrawableRectangle extends Rectangle DrawableShape { . . . }
```

which is illegal in Java.

So we introduce an interface called Drawable::

```
public interface Drawable {
        public void setColor(Color c);
        public void setPosition(double x, double y);
        public void draw(DrawWindow dw);
```

```
}
public class DrawableRectangle extends Rectangle implements Drawable {
        private protected Color c;
        private protected double x, y;   // position
        DrawableRectangle(double w, double h) {super(w, h);}

        // define abstract methods of Drawable interface
        public void setColor(Color c) {this.c = c;}
        public void setPosition(double x, double y) {
                this.x = x;
                this.y = y;
        }
        public void draw(DrawWindow dw) {
                dw.drawRect(x, y, w, h, c);
        }
}
```

And now we can use the above definitions as follows:

```
Shape[ ] shapes = new Shape[2];
·Drawable[ ] drawables = new Drawable[2];

shapes[0] = drawables[0] = new DrawableCircle(1.1);
shapes[1] = drawables[1] = new DrawableRectangle(2.3, 4.5);

double total_area = 0;
for (int i = 0; i < shapes.length; i++) {
        total_area += shapes[i].area( );         // dynamic dispatch
        drawables[i].setPosition(i * 10, i * 10);
        drawables[i].draw(draw_window); //draw_window defined elsewhere
}
```

Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw.html

# Packages

In Java, class definitions are organized using **packages**.

Every class belongs to a package.

The package to which it belongs is specified by a **package** declaration:

**package** pkg1 [ . pkg2 [ . pkgn ] ];

For example, we declare class **Hashtable** to be in package java.util by

```
package java.util;
public class Hashtable extends Dictionary implements Cloneable {
        // see page 343 of Nutshell
}
```

A package is merely a directory, and the "dots" indicate subdirectories.

On UNIX systems, **Hashtable** must be in directory java/util.

On Windows/NT platforms, **Hashtable** is in java\util.

java.util is a **relative** path that is resolved with respect to a platform-dependent default path, for example, c:\java\classes. This is true of system classes, i.e. those in the package "java".

User-defined packages are resolved with respect to the CLASSPATH environment variable.

```
package Sets.Heap;
class Heap { . . . }

package Sets.BitVector;
class BitVector { . . . }
```

If CLASSPATH is ".;c:\myjava", then class Heap must be in c:\myjava\Sets\Heap and BitVector in c:\myjava\Sets\BitVector.

To use these classes, one writes

```
Sets.Heap s1 = new Sets.Heap( );
Sets.BitVector s2 = new Sets.BitVector( );
```

Java permits path names to be abbreviated using **import**:

```
import Sets.Heap;
import Sets.Bitvector;
Heap s1 = new Heap( );
BitVector s2 = new Bitvector( );
```

which can be abbreviated still further as

```
import Sets.*;
Heap s1 = new Heap( );
BitVector s2 = new Bitvector( );
```

In general, a class path pkg1.pkg2.pkgn.classname can be written as just classname providing there is an import statement of the form

**import** pkg1.pkg2.pkgn.classname

**import** declarations are merely a notational shorthand. They are NOT needed.

# Access protection

Java provides control over the visibility of variables and methods.

Control is exercised at three levels:

1. classes
2. subclasses
3. packages

See the table on page 134 of the Java Handbook.

The places from which one wishes to "see" definitions are

1. **same class**

   ```
   class C {
           int i;
           int f( ) { return i;}   // can f see i ?
   }
   ```

2. **same package subclass**

   ```
   // in a source file
   package p;
   class C {
           int i;
   }

   // in another source file
   package p;
   class D extends C {
           int f( ) {return i;}              // can f see i ?
   }
   ```

3. **same package non-subclass**

   ```
   // in a source file
   package p;
   class C {
   ```

```
        int i;
}
// in another source file
package p;
class D {
        int f( ) {return i;}              // can f see i ?
}
```

## 4. **different package subclass**

```
// in a source file
package p;
class C {
        int i;
}

// in another source file
package q;
class D extends C {
        int f( ) {return i;}              // can f see i ?
}
```

## 5. **different package non-subclass**

```
// in a source file
package p;
class C {
        int i;
}
// in another source file
package q;
class D {
        int f( ) {return i;}              // can f see i ?
}
```

The access modifiers are

1. **private** - visible in class only
2. **public** - visible everywhere
3. **protected** - visible in package and subclasses
4. **private protected** - visible in subclasses
5. the default - visible in package

See the example in Protection.java and Protection2.java.

Index


this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw.html

*java@nps.navy.mil*
Last Updated 25 JUL96
Naval Postgraduate School © 1996

# File & FilenameFilter classes

## File

The File object enables the assignment of a handle via the java.io package by which the programmer can access information about actual physical files. It can access certain properties of the files and the file system but does not deal with how information is stored, that is left to the operating system.

As in Unix, Java treats directories as files that have lists of filenames in them. Regular files have many properties that the Java File object can access thru methods including existence, readability, size, type and time of creation.
(See Nutshell p. 296 for a complete description)

The following code (from p. 221 of Handbook) demonstrates some of the properties of a File object.

```java
import java.io.*;
class FileTest {

    private static void p(String s) {
        System.out.println(s);

    }
    public static void main(String args[]) {

        File f1 = new File("/java");

        java.util.Date d = new java.util.Date(f1.lastModified()); //for fun

        p("File Name:" + f1.getName());
        p("Path:" + f1.getPath());
        p("Platform Path Separator: "+ f1.separator);
        p("Current directory: "+ System.getProperty("user.dir"));
        p("Abs Path:" + f1.getAbsolutePath());
        p("Parent:" + f1.getParent());
        p(f1.exists() ? "exists" : "does not exist");
        p(f1.canWrite() ? "is writeable" : "is not writeable");
        p(f1.canRead() ? "is readable" : "is not readable");
        p("is " + (f1.isDirectory() ? "" : "not ") + "a directory");
        p(f1.isFile() ? "is normal file" : "might be a named pipe");
        p(f1.isAbsolute() ? "is absolute" : "is not absolute");
        p("File last modified:" + f1.lastModified());
        p("File last modified:" + d.toString());
        p("File last modified:" + d.toLocaleString());
        p("File size:" + f1.length() + " Bytes");
    }
}
```

Note: enhanced and corrected from text.
Sample: FileTest.java

Sample: Ren.java renames an existing file.

---

# FilenameFilter

This class filters out filenames based on filename pattern matching. This can also work on the read or write flags of a file. It uses the `accept` method which is called on every item in the list. This interface is implemented in the AWT FileDialog object and the `list` method of `File`.
(From Handbook, p.222-223)
The authors DirList.java
Sample: DirList.java
Sample: OnlyExt.java

this page is located at
http://vislab-www.nps.navy.mil/~java/course/filefilter.html

*java@nps.navy.mil*
Last Updated 30 JUL96
Naval Postgraduate School © 1996

# Streams Overview

A stream is an abstract concept that refers to the movement of data from one point to another. We typically think of this data as a sequence of bytes which we can either read or write. The bytes are frequently read from or written to a file, but other devices such as network connections or memory blocks are handled in essentially the same way.

## Two basic abstract types:
## InputStream and OutputStream

InputStream is the superclass of all input streams. These methods will throw an `IOException` on error. An input stream is automatically opened when you create it. Each subclass has

- `read()` reads the next byte of data on the stream.
- `read(byte b[])` reads into an array of bytes, returns number read.
- `read(byte b[], int off, int len)` starts at offset, reads into an array.
  all `read()` methods return -1 at the end of the stream.
- `skip (long n)` skips n bytes and returns actual number skipped.
- `available()` returns number of bytes available without blocking.(threads)
- `close()` closes the stream and frees resources.
- `markSupported()` returns `true` if supported.
- `mark (int readlimit)` marks current position in the stream.
- `reset()` returns stream pointer to last`mark`position.

OutputStream provides the basic output methods for all subclasses. An output stream is automatically opened when you create it.These methods all return `void` and throw `IOException` on error.

- `write(int b)` writes a single byte to the stream.
- `write(byte b[])` writes an entire array of bytes to the stream.
- `write(byte b[], int off, int len)` writes a subarray to the output stream.
- `close()` closes the output stream.
- `flush()` clears the output stream.

Beneath these abstract classes are more refined stream types:

- File Streams
- Filtered Streams:
  - O Print Streams
  - O Data Streams
  - O Buffered Streams
- SequenceInputStream
- RandomAccessFile

- Piped Streams

File Streams:
> The FileInputStream and FileOutputStream classes are used to read and write data to actual files in a host system.

**FilterInputStream** and **FilterOutputStream** are subclasses of InputStream and OutputStream, respectively, and are both themselves abstract classes. These classes define the interface for filtered streams. Filtered streams process the data as its being read or written.

Print Streams:
> Extends FilterOutputStream to print textual representations of all primitve data types. Uses `print()` and `println()` to output the text. `Println()` follows the text with a newline. An `Object` is converted with its `toString` method before printing. This method is used by the `System.out` variable. Note that this class may not print all Unicode characters

Data Streams:
> Used to read and write many primitive Java types from or to the respective stream in platform independent binary format. DataOutputStream does not handle Strings. Example of keyboard echo.

Buffered Streams:
> Provide data buffering to increase efficiency. When writing data to a physical device or memory block, multiple accesses may take longer than waiting for the buffer to fill, then going to the disk. The size of the buffer may be specified, else it defaults to 32 bytes. By setting up a such an input stream, an application can read bytes from a stream without necessarily causing a call to the underlying system for each byte read. The data is read by blocks into a buffer; subsequent reads can access the data directly from the buffer.
>
> Example of code using streams:
>
> ```
> FileInputStream in = null;
> DataInputStream dataIn = null;
> BufferedInputStream bis = null;
>
> in = new FileInputStream(args[1]);
> bis = new BufferedInputStream(in);
> dataIn = new DataInputStream(bis);
> ```
>
> OR
>
> ```
>    DataInputStream dataIn =
> new DataInputStream(new BufferedInputStream(new FileInputStream(args[1])));
> ```

SequenceInputStream
> Creates a single input stream on multiple input sources.This example concatenates two files to the console: Concatenate.java uses ListOfFiles.java.

RandomAccessFile:
> Though not a stream at all, this class implements DataInput and DataOutput and allows the reading

and writing of any mixture of datatypes to any location in a file. This object has a mode associate with it, either "r" or "rw" for read only or read-write. A physical filename or object is passed in the constructor.

Piped Streams:
  Provide methods for allowing two threads or programs to send data between them in a reliable manner. A PipedInputStream must be connected to a PipedOutputStream and vice versa.

# Review

- File Objects: provides handle to a physical file and returns properties of files.
- FilenameFilter: boolean accept method, matches file properties or names.
- InputStreams: basic methods for read, skip, mark, etc bytes of data.
- OutputStream: returns void, write, flush, and close methods.
- FileStreams: uses File object to read and write data to files
- Filered streams: process the data as its being read or written.
- PrintStream: to standard output, uses print or println
- DataStreams: primitive types, platform independant binary.
- Buffered Streams: increase efficiency.
- SequenceInputStream: connects multiple input streams sequentially.

From Nutshell p. 131:FileCopy.java

# FileInputStream

This InputStream subclass is used to attach a "handle" to a physical sytem file for reading. It provides a low-level interface and is usually used with a `DataInputStream` to provide a higher-level interface for reading data and text. The public constructor can take a filename, `File` object or `FileDescriptor` object.

(This is from Nutshell p. 126)
Sample: Fileviewer.java

# FileOutputStream

This InputStream subclass is used to attach a "handle" to a physical sytem file for reading. The public constructor can take a filename, `File` object or `FileDescriptor` object. If the file does not already exist, it will be created.

# Streams Overview

A stream is an abstract concept that refers to the movement of data from one point to another. We typically think of this data as a sequence of bytes which we can either read or write. The bytes are frequently read from or written to a file, but other devices such as network connections or memory blocks are handled in essentially the same way.

## Two basic abstract types:
## InputStream and OutputStream

InputStream is the superclass of all input streams. These methods will throw an `IOException` on error. An input stream is automatically opened when you create it. Each subclass has

- `read()` reads the next byte of data on the stream.
- `read(byte b[])` reads into an array of bytes, returns number read.
- `read(byte b[], int off, int len)` starts at offset, reads into an array.
  all `read()` methods return -1 at the end of the stream.
- `skip (long n)` skips n bytes and returns actual number skipped.
- `available()` returns number of bytes available without blocking.(threads)
- `close()` closes the stream and frees resources.
- `markSupported()` returns `true` if supported.
- `mark (int readlimit)` marks current position in the stream.
- `reset()` returns stream pointer to lastmarkposition.

OutputStream provides the basic output methods for all subclasses. An output stream is automatically opened when you create it.These methods all return `void` and throw `IOException` on error.

- `write(int b)` writes a single byte to the stream.
- `write(byte b[])` writes an entire array of bytes to the stream.
- `write(byte b[], int off, int len)` writes a subarray to the output stream.
- `close()` closes the output stream.
- `flush()` clears the output stream.

Beneath these abstract classes are more refined stream types:

- File Streams
- Filtered Streams:
  - Print Streams
  - Data Streams
  - Buffered Streams
- SequenceInputStream
- RandomAccessFile

- Piped Streams

File Streams:
The FileInputStream and FileOutputStream classes are used to read and write data to actual files in a host system.

**FilterInputStream** and **FilterOutputStream** are subclasses of InputStream and OutputStream, respectively, and are both themselves abstract classes. These classes define the interface for filtered streams. Filtered streams process the data as its being read or written.

Print Streams:
Extends FilterOutputStream to print textual representations of all primitve data types. Uses `print()` and `println()` to output the text. `Println()` follows the text with a newline. An `Object` is converted with its `toString` method before printing. This method is used by the `System.out` variable. Note that this class may not print all Unicode characters

Data Streams:
Used to read and write many primitive Java types from or to the respective stream in platform independent binary format. DataOutputStream does not handle Strings. Example of keyboard echo.

Buffered Streams:
Provide data buffering to increase efficiency. When writing data to a physical device or memory block, multiple accesses may take longer than waiting for the buffer to fill, then going to the disk. The size of the buffer may be specified, else it defaults to 32 bytes. By setting up a such an input stream, an application can read bytes from a stream without necessarily causing a call to the underlying system for each byte read. The data is read by blocks into a buffer; subsequent reads can access the data directly from the buffer.

Example of code using streams:

```
FileInputStream in = null;
DataInputStream dataIn = null;
BufferedInputStream bis = null;

in = new FileInputStream(args[1]);
bis = new BufferedInputStream(in);
dataIn = new DataInputStream(bis);
```

OR

```
    DataInputStream dataIn =
new DataInputStream(new BufferedInputStream(new FileInputStream(args[1])));
```

SequenceInputStream
Creates a single input stream on multiple input sources.This example concatenates two files to the console: Concatenate.java uses ListOfFiles.java.

RandomAccessFile:
Though not a stream at all, this class implements DataInput and DataOutput and allows the reading

and writing of any mixture of datatypes to any location in a file. This object has a mode associate with it, either "r" or "rw" for read only or read-write. A physical filename or object is passed in the constructor.

Piped Streams:
> Provide methods for allowing two threads or programs to send data between them in a reliable manner. A PipedInputStream must be connected to a PipedOutputStream and vice versa.

# Review

- File Objects: provides handle to a physical file and returns properties of files.
- FilenameFilter: boolean accept method, matches file properties or names.
- InputStreams: basic methods for read, skip, mark, etc bytes of data.
- OutputStream: returns void, write, flush, and close methods.
- FileStreams: uses File object to read and write data to files
- Filered streams: process the data as its being read or written.
- PrintStream: to standard output, uses print or println
- DataStreams: primitive types, platform independant binary.
- Buffered Streams: increase efficiency.
- SequenceInputStream: connects multiple input streams sequentially.

From Nutshell p. 131:FileCopy.java

# FileInputStream

This InputStream subclass is used to attach a "handle" to a physical sytem file for reading. It provides a low-level interface and is usually used with a `DataInputStream` to provide a higher-level interface for reading data and text. The public constructor can take a filename, `File` object or `FileDescriptor` object.

(This is from Nutshell p. 126)
Sample: Fileviewer.java

# FileOutputStream

This InputStream subclass is used to attach a "handle" to a physical sytem file for reading. The public constructor can take a filename, `File` object or `FileDescriptor` object. If the file does not already exist, it will be created.

---

this page is located at
http://vislab-www.nps.navy.mil/~java/course/streams.html

*java@nps.navy.mil*
Last Updated 30 JUL96

# Applets and Applications

---

## Applications

We have already discussed how the `main()` method works and is required for starting applications that typically run from the command line with the Java interpreter. To this point, our graphical applets have run in either in the appletviewer or on web pages in a browser. As we begin to explore the AWT, we will also see programs, not applets, that have a graphical interface yet are invoked with the Java interpreter and run without the aid of a viewer.

Example:Win_Clock.java Seen here as an Applet.

## Applets

We have previously covered how to embed applets in web pages with the `<applet>` and `<param>` tags (HW1). Now we will look at the pieces that make the applets work.

Every serious applet uses these four methods found in the `java.applet.Applet` class: `init()`, `start()`, `stop()`, `destroy()`. The Applet class has these default methods in place, however they are empty so it is common to override them to enhance their functionality.

**init**
This method is used for whatever initializations are needed in your applet. It works much like a constructor since it is automatically called by the system when Java launches the applet for the first time. Common things to do in the init include initailizing variables, getting items passed via the `<param>` values and building the user-interface components.

Lets look at the ColorScribble init method.

**start**
This is the second method automatically called after `init` and is also called whenever a `stop` has been called or the user has left the page and then later returned to it. This is where the work gets done. It is here that you would start a thread of execution. Therefore, if it only has to run once, put it in the `init`, otherwise it goes in `start`. If you are not doing anything that needs to be suspended, this method and the `stop` method do not need to be implemented.

**stop**
Stops the applet from executing. This method is called automaticaly when the user leaves the page where the applet resides. It can also be used to suspend threads that don't need to be running when the applet is not visible or to stop a resource consuming activity that may be slowing the system down. You usually will not need to call this method directlyc unless you are performing heavy calculations or running audio or animations.

**destroy**
Frees up any system resources that the applet is holding. If the applet is running in a browser, the browser will determine when to call `destroy`. Java is guaranteed to call this method when the browser shuts down normally.

Two other methods that show up frequently in graphical applets are `paint` and `update`.

**paint**
This method will run anytime the applet is damaged, usually as a result of other system windows, pop-up dialog boxes or resizing of the applet window. It gets called frequently.

**update**
Each time the window needs to be redrawn, for whatever reason, the eventhandler calls its `update` function. The default implementation for update, defined in `Component`, is to erase the background and call `paint`. To avoid a flash when repainting,, the `update` method may be overridden.(Some authors prefer to override `paint` instead.)

A Demo by Gordon Bradley that illustrates how these all relate.

A word about `System.getProperty()` p.324, 194


this page is located at
http://vislab-www.nps.navy.mil/~java/course/appsapps.html

*java@nps.navy.mil*
Last Updated 30 JUL96
Naval Postgraduate School © 1996

# Applets and Applications

---

## Events

Building graphical intefaces with buttons, shapes, lines, menus and checkboxes is a fairly simple task. The functionality of any applet can only be realized by incorporating **event handlers** for the objects in use. The event type values are defined by the `java.awt.Event` class and are stored in the `id` field of the `Event` object.

GUIs and Windowing systems use event-driven models rather than procedural- grab- the- input-yourself models. The O/S polls constantly for events. When an event is located, it is relayed to the program and the program decides how to handle that event.

An `Event` object is created anytime a user does something, like clicking a button or pressing a key, on a `Component`. The `Component` `deliverEvent()` method takes the `Event` object from the top most component, like the browser or a `Window`, and sends it down to the lowest `Component`, like a button.

Once the target component is reached, the AWT allows each `Component` to handle the event at every step of the `Component` hierarchy. At each component level, the Event handler may modify the `Event` instance before it is passed up, stop the Event from being processed further or react in some other way by doing interim processing.

Each `Event` object includes the following information:

1. The type of the event(a key press or mouse click).
2. The object that was the "target" of the event(the Button clicked).
3. A timestamp indicating when the event occurred.
4. The location (x,y) where the event occurred.
5. The key that was pressed (for keyboard events).
6. An arbitrary argument (such as the string displayed on the Component)
7. The state of the modifier keys when the event occurred.

Unhandled events are passed to the parent container via `postEvent()` so they get a chance to deal with the event. `Components` can respond to events by implementing a custom written `handleEvent()` method or by using the default (`Component`) definition of `handleEvent()` which simply calls a method that's specific to the event type.

Events that routinely need to be handled.
(1) **ACTION_EVENT** - Occurs when you press a button, select a menu, etc.
(2) **KEY_ACTION** - Occurs when text field action occurs.
(3) **MOUSE_DOWN** - Occurs when the mouse button is clicked down.
(4) **WINDOW_DESTROY** - Occurs when the window of the object is closed.

A sample:

```
class ClosableFrame extends Frame
{
        public ClosableFrame(String t)
        {
                super(t);
        }

        public boolean handleEvent(Event e)
        {
                if (e.id == Event.WINDOW_DESTROY) System.exit(0);
                return super.handleEvent(e);
        }
}
```

A full listing of the Event types is listed on p. 183-185 of Nutshell

Now how does it work? Some Examples please!

Look at the EventTester on p90 of Nutshell.

Notice the `init()` method and the `getParameter()` method of ColorScribble p. 92

If you want to have really easy to use interfaces, combine mouse events with keyboard events so that, for example, a user can either click on a button or tab to get the forcus then press enter. (find example).


this page is located at
http://vislab-www.nps.navy.mil/~java/course/events.html

*java@nps.navy.mil*
Last Updated 30 JUL96
Naval Postgraduate School © 1996

# Components and Containers
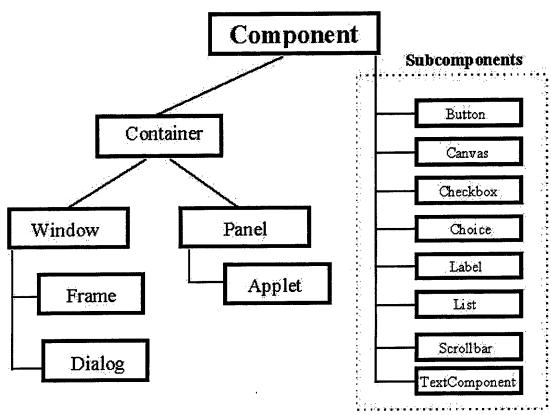


Figure 2 Structure of AWT

## The `java.awt.Component` Class

This abstract class contains the basic methods that all its subclassed components can use. Components are the window icons with which the user interacts - Buttons, TextAreas, Scrollbars, etc. Basically, Components are the visible GUI controls added to a `Container`. Anything derived from the `java.awt.Component` class can be a `Component`. (Nutshell p.244)

Some common methods used in the `java.awt.Component` package are:

- `getBackground()`
- `setBackground(Color)`
- `getFont()`
- `setFont(Font)`
- `mouseDown(Event, int, int)`
- `show()`
- `resize(int, int)`
- `paint(Graphics)`
- `update(Graphics)`

- `move(int, int)`

# Containers

The `Container` class is an abstract subclass of `Component` and, is designed to group together related components. A `Container` may also contain other `Components`. The two subclasses are `Windows`, which provide the windows to contain `Components`, and `Panels`, which group `Components` within the area of an exiting window. In general, `Components` must be added to a `Container` before they can be used or modified. They use the `add()` method with one or two arguments depending on the `LayoutManager` used. Common methods used in `Containers` are:

```
add(Component);
add(String, Component);
remove(Component);
getComponents();
setLayout(LayoutManager);
getLayout();
```

# Windows

When an application is written for a GUI using the AWT, the Window class provides the framework for a basic standalone process. The Window is the next major subclass of Container that provides a top level window with no borders or menu bar. Although it is rarely called directly, it does provide the important event handles `WINDOW_DESTROY`, `WINDOW_ICONIFY`, `WINDOW_DEICONIFY` and `WINDOW_MOVE`. The two Window subclasses are `Frame` and `Dialog`.

## Frame

The `Frame` subclass of `Window` provides a container for all associated components used in the main window of the program. The `Frame` has many predefined convenience methods and may have a specified title, menu bar, icon and cursor. The `Frame` does not need to be displayed physically within any other type of container or component, it can stand freely on its own. The `show()` and `hide()` methods make the `Frame` appear and disappear when needed. The `setResizable(boolean)` determines whether the user can resize the `Frame`. `Frame` is the **only** container class supporting the `MenuComponents` class and subclasses.

The `MenuComponent` class is extended from the `Object` class, so it is not a component that can be usable anywhere or instantiated directly. The subclasses using `MenuComponent` are `MenuBar, Menu, MenuItem and CheckboxMenuItem`. The `MenuBar` sets up the platform-dependent group of menus, typically across the top of the window. Each menu and the drop-down list of options is represented by a `Menu` object. Each object on the list is a `MenuItem`. `Menu` is implemented as a subclass of `MenuItem`, so sub-menus can be made by adding one menu to another. For example, `CheckboxMenuItem` is a subclass of `MenuItem` class.

## Dialog

The `Dialog` class provides a simple, encapsulated dialog box which blocks further user input until dismissed. `Dialogs` are dependent on other windows. If the parent window is iconified or destroyed, the

`Dialog` box goes with it. A handy subclass of `Dialog` is the `FileDialog` which brings up the native file chooser window to allow local file selection. `FileDialog` currently has Load and Save buttons built in, but other operations can also be supported.

```
Dialog(Frame parent, boolean modal)  no title on window.
Dialog(Frame parent, String title, boolean modal)
FileDialog(Frame parent, String title)
```

Index


this page is located at
http://vislab-www.nps.navy.mil/~java/course/component.html

*java@nps.navy.mil*
Last Updated 19 AUG 96
Naval Postgraduate School © 1996

# Subcomponents

Subcomponents and their constructors are added using the add(Component) or addItem(String) method and removed with remove(). Adding a component doesn't necessarily create the native Subcomponent (peer Subcomponent) for that component, but simply inserts the component into the component hierarchy. The native Subcomponent only gets instantiated when pack() or show() is called on a window else when you add a component to a container whose native subcomponent is already created. The Subcomponents provide basic controls, labels, and methods for creating custom components for all types of GUI interfaces. Many trigger ACTION_EVENTs for interactive applications.

The Button creates a simple on-screen button the user can push. A string within the constructor shows up as a label on the button.

```
Button();
Button(String);
```

The Canvas provides a semantic-free, nearly abstract class which inherits its functionality from the Component class. The Canvas is well suited for building custom components. It is ready for use as a drawing surface and allows the implementation of event handling. When the Canvas is extended, inherited methods are overwritten to provide the desired functionality.

The Checkbox has a selected/deselected toggle control and may also have a label. It may be initialized with either true or false boolean value. A series of checkboxes in the same CheckboxGroup will be mutually exclusive.

```
Checkbox();
Checkbox(String);
Checkbox(String, CheckboxGroup, Boolean);
```

The Choice creates a pop-up menu on the screen when the user clicks on it. Items are listed in the order added to the class with addItem. Only the first item added will be displayed on top.

```
Choice C = new Choice();
C.addItem("First choice");
```

The Label displays a static line of text, although it may be constructed empty and set at a later time. Labels also have an alignment property of LEFT, CENTER or RIGHT stored as an integer.

```
Label();
Label(String);
Label(String, int);
```

The List displays a scrolling list of items of which one or many may be selected. A List can specify the number of lines to be visible or use the size of the container as the default. A boolean is set to determine

if multiple selections are permitted.

```
List   itemlist  = new List(1, true);
itemlist.addItem("first item");
itemlist.addItem("second item");
```

The `Scrollbar` displays the familiar windows control for moving through a range of values. The orientation is set to HORIZONTAL or VERTICAL. The initial value or position of the *thumb* is set to an integer value. The visibility or overall area displayed is specified in pixels. The minimum and maximum size of the scrollable area are also specified in pixels as integers.

```
Scrollbar(orientation);
Scrollbar(orientation int, initial_value int, visible
      int, minimum int, maximum int);
```

The `TextComponent` superclass of `TextArea` and `TextField` supplies many methods which allow the programmer to set the text editable or not, get the text after user input and select text between beginning and end positions the programmer specifies. The `TextField` creates field with a single, editable line of text. The number of columns in the text field may be set with an integer, and the text initially displayed is set with a String. `TextField` also has the useful `setEchoChar()` method which is useful for password fields.

```
TextField();
TextField(int);
TextField(String);
TextField(String, int);
```

`TextArea` creates a multi-line box for text entry. The `TextArea` may be any height and width-scrollbars are added by default. The int arguments set the number of rows and columns (respectively) of the area. The `appendText()` method allows text to be added to the `TextArea` incrementally. Other methods include `insertText(String, int)` and `replaceText(String, int, int)`. Like the `TextField` class, `TextArea` has four constructors.

```
TextArea();
TextArea(int, int);
TextArea(String);
TextArea(String, int, int);
```

Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/subcomps.html

*java@nps.navy.mil*
Last Updated 19 AUG 96
Naval Postgraduate School © 1996

# Panels and Layout Managers

## Panel

A `Panel` is one of the most generic, yet concrete, containers available which can be displayed on-screen. Once a component is added to a `Panel`, it can be manipulated using the `move()`, `resize()` or `reshape()` methods inherited from `Component`. Each `Panel` or sub-panel may be treated as a `Container` in its own right. To offer more control and flexibility in how components are displayed within a container, the abstract class `LayoutManager` is used. The final appearance of a component is determined by which `LayoutManager` is used, and the order in which items are added to the container.

## Layout Managers

Layouts define how Components are "laid out" within a Container. There are five predefined subclasses of the LayoutManager. Since the LayoutManager class is abstract, it cannot be used directly. Instead, it must be a sub-class with its own functionality or, use a derived class defined in the AWT - i.e. BorderLayout, CardLayout, GridLayout, FlowLayout, etc. The actual appearance of the AWT components on-screen is determined by the order in which they are added to the Panel. (and the LayoutManager class the panel is currently using to lay out the screen.) The LayoutManager class determines how portions of the screen are sectioned and how components within the Panel are placed. Layouts at Sun.

The **BorderLayout** arranges components around the edges of the container according to an area specified in the instantiation of the component. The areas of choice are: north, south, east, west, and center. The area in which components are added is important. If a center component is added it gets allocated all remaining space. The other areas expand only as much as necessary to keep all available space filled. The argument add() method is used to add a component with BorderLayout. If a gap is desired between components, the horizontal and vertical pixel values are set with the setLayout method. BorderLayout is the default layout manager for all Windows, such as Frames and Dialogs.

```
        Add("direction", new Component("string"));
BorderLayout(int horizontalGap, int verticalGap);
```

The **CardLayout** gets its name because only one visible display space is visible although many Containers may use the same space. Like a stack of playing "cards", components may be displayed by calling the methods first(), last(), next(), previous(), or show() to get a specific card. Again, the argument add(String name, Component comp) is required with the String being an identifier for that component. The order in which components are added determines their order in the "deck".

```
        Panel card1 = new Panel();
        card1.setLayout(new CardLayout());
        add("first", card1);
        card1.show(card1,"first");
```

The **GridLayout** provides an easy method for organizing components in a grid. All components are placed in equally sized cells, and placed in the container in the order in which they are added, starting in the top row from left to right. If a window containing a GridLayout is resized, the cells grow or shrink so that all the space available to the container is used. The GridLayout constructor includes the number of rows and columns required. As an option, horizontal and vertical gaps maybe specified in pixels. At least one of the row arguments or column arguments must be non-zero.

```
     GridLayout(int rows, int columns);
GridLayout(int rows, int columns, int horizontalGap, int verticalGap);
```

The **GridBagLayout** is much more flexible than GridLayout as it allows specific components to span multiple rows or columns. The characteristics of each component is optionally set with GridBagConstraints. Various options may be set including anchor position; grid position, which may be relative or absolute; the number of horizontal and vertical cells occupied; fill direction; margin insets; and directional weights.

```
GridBagLayout();
```

Sun's GridBagConstraints lesson.

The **FlowLayout** is probably the simplest to use. Its puts components in a row until the horizontal space is full, then shifts to multiple lines. Horizontal and vertical gaps may be specified with a default of five pixels. The alignment of the components on the rows may be LEFT, RIGHT, or CENTER justified when the FlowLayout is created. FlowLayout is the default layout manager for all Panels.

```
FlowLayout();
FlowLayout(int alignment);
FlowLayout(int alignment, int horizontalGap, int verticalGap);
```

A great set of examples from Sun.

Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/subcomps.html

*java@nps.navy.mil*
Last Updated 19 AUG 96
Naval Postgraduate School © 1996

# Networking

`java.net` supports both TCP/IP and UDP protocol families.

TCP/IP is used for reliable stream-based communication across the Internet. It specifies the manner in which two processes running on different machines on the Internet find each other, rendezvous, and transfer data. It also makes sure data are transferred in the correct sequence and without error.

UDP (Unreliable Datagram Protocol) is used for unreliable datagram communication; also called "fire-and-forget" packet transmission across the network.

TCP/IP uses a static addessing scheme--every machine on the Internet is assigned a unique, fixed *IP address*.

# IP Addresses

IP addresses are 32-bit numbers usually written as four 8-bit numbers separated by dots.

For example, 18.157.0.135 and 127.1.18.92.

IP addresses are organized in a hierarchical way into a series of networks and subnets. The Network Information Center (NIC) allocates blocks of contiguous addresses to organizations and regional networks.

A small organization might receive the block of 255 addresses from 192.66.12.1 to 192.66.12.255 (this is called a class "C" address).

A large organization, such as a university, might receive the block of approximately 65,000 addresses from 128.15.0.1 to 128.15.255.255 (this is called a class "B" address).

Even larger entities, such as the military or a regional net, might be granted one or more class "A" addresses, such as the block 18.0.0.1 to 18.255.255.255, encompassing more than 16 million addresses.

Longer addresses will replace the current ones over the next few years.

## Networks and Hosts

```
Class   Address         Network Part    Host Part
A       18.155.32.5     18.             155.32.5
B       128.15.32.5     128.15          32.5
C       192.66.12.56    192.66.12       56
```

Raw IP addresses are assigned names using a distributed, hierarchical lookup system called the Domain Name System (DNS). In DNS, each machine has a unique name consisting of multiple parts separated by dots. The first part is the machine's host name, followed by a list of *domains*.

The first domain is usually an identifier for the organization to which the machine belongs, followed by more subtitles, and concluding with a label for the *top-level* domain. In the U.S., the top-level domain is usually edu (educational), com (commerce), mil (military), net (network providers), and org (other organizations).

For the rest of the world, the top-level domain usually identifies the country: jp (Japan), de (Germany or Deutschland), and so on.

The dots in domain names have no correspondence to the dots in IP addresses. Domain names may have two, three, or more, depending on how the local naming system is set up. For example,

gold.cs.nps.navy.mil

has five parts. The top-level domain is mil for military.

The information in the DNS system is distributed among a large number of DNS databases called domain name servers, each maintained by the organization responsible for its piece of the network. The DNS databases are just human-readable tables associating numeric IP addresses with host names.

When a program, like a browser or rlogin, is given a domain name to connect to, it first queries its local domain name server (in the CS dept, it is machine taurus) to find the numeric IP address for the name. If the server doesn't know, which is often the case, it queries another name server closer to the destination, and that server may in turn query a third, and so on.

# Ports

When two network programs wish to communicate, it is not enough for them each to know the other's IP address. A single machine may be running multiple programs as *servers*. For a particular program to be able to communicate with a server, the program needs to know to which **port** the server is "connected". That is, to which port is the server "listening"? Ports are soft (operating system entities). On UNIX systems, when a server starts up, it notifies the operating system of the port it will listen to.

For example, a typical UNIX machine offers TELNET and FTP servers. It may also be running a Web server like the CERN server or NCSA's httpd. On UNIX systems, port numbers between 0 and 1024 are reserved, by convention, for particular servers:

```
Protocol          Port
date server       13
FTP               21
Telnet            23
SMTP              25
Gopher            70
HTTP              80
Usenet news       119
WAIS              210
```

So if a network program wishes to communicate with the Web server, it must know to which port the server is connected.

# Networking in Java

## The InetAddress Class

The class represents an Internet address. It has no public constructors, just three class methods that create instances of the class:

1. InetAddress **getLocalHost( )** throws UnknownHostException;
2. InetAddress **getByName**(String *host*) throws UnknownHostException;
3. InetAddress[] **getAllByName**(String *host*) throws UnknownHostException;

The instance methods include:

1. byte[] **getAddress( )**;
2. String **getHostName( )**;

getAddress( ) returns the IP address as an array of bytes in "network byte order"; getHostName( ) returns the host name of an IP address.

See InetDemo.java.

## Datagrams

Datagram communication by UDP is fast, but the tradeoff is that

1. the protocol makes no attempt to ensure they reach their destination or to resend them if they did not
2. they are not stream based; there is no semi-permanent connection
3. they are unidirectional

To send and receive datagrams in Java, use the DatagramPacket and DatagramSocket classes.

## DatagramPacket

The DatagramPacket class has two constructors:

1. **DatagramPacket**(byte [], *buf*, int *buflength*);
2. **DatagramPacket**(byte [], *buf*, int *buflength*, InetAddress *ipaddr*, int *port*);

The first constructor is used to receive a datagram while the second is used to send a datagram.

The contents and sender of a received packet may be queried with the instance methods:

1. InetAddress **getAddress( )**;
2. byte[] **getData( )**;
3. int **getLength( )**;

4. int **getPort( )**;

Datagram packets are sent and received using the DatagramSocket class.

## The DatagramSocket Class

The class defines a socket that can receive and send unreliable datagram packets over the network.

The class has two constructors:

1. **DatagramSocket( )** throws SocketException;
2. **DatagramSocket**(int *port*) throws SocketException;

The first constructor is used to create a datagram socket for sending a packet and the port to be used is chosen by the system.

The second constructor creates a socket for sending or receiving packets using the instance methods:

1. void **send**(DatagramPacket *p*) throws IOException;
2. void **receive**(DatagramPacket *p*) throws IOException;

See WriteServer.java and ReadClient.java.

# Sockets for Clients

TCP/IP sockets are used to implement reliable bi-directional, stream-based connections between a server and a client on the Internet.

Java's I/O system can be connected to a server so that streams from files and servers are treated uniformly.

In Java, TCP/IP sockets are created via the class **Socket**.

## The Socket Class

The Socket class has four constructors:

1. **Socket**(String *host*, int *port*) throws UnknownHostException, IOException;
2. **Socket**(String *host*, int *port*, boolean *stream*) throws IOException;
3. **Socket**(InetAddress *addr*, int *port*) throws IOException;
4. **Socket**(InetAddress *addr*, int *port*, boolean *stream*) throws IOException;

You may optionally specify whether communication through the socket should be based on a connection-based stream protocol or UDP.

A socket can be examined for the address and port associated with it using:

1. **getInetAddress( )** returns the InetAddress of the Socket object.
2. **getPort( )** returns the *remote* port of the Socket object.
3. **getLocalPort( )** returns the local port of the Socket object.

Unlike datagram sockets, client sockets are handled using stream-based I/O classes in Java, just as files are handled:

1. **getInputStream( )** returns the InputStream of the Socket object.
2. **getOutputStream( )** returns the OutputStream of the Socket object.
3. **close( )** closes both the InputStream and OutputStream.

See the following examples:

1. QueryServer.java
2. QueryHTTP.java

# Sockets for Servers

A server socket "listens" to a port for connection requests from clients.

When a connection request arrives, a ServerSocket object creates a new Socket object for communication with the client.

ServerSocket has two constructors:

1. **ServerSocket**(int *port*) throws IOException;
2. **ServerSocket**(int *port*, int *count*) throws IOException;

The first creates a server socket on the specified port. The second creates a server socket on the specified port, waiting up to *count* milliseconds for a connection request.

ServerSocket has the following instance methods:

1. Socket **accept( )** throws IOException; blocks until a connection request is detected from a client at which time it returns a new Socket object for communication with the client.
2. InetAddress **getInetAddress( )**; returns the InetAddress of the client issuing the connection request.
3. **getLocalPort( )**; returns the local port in use by the ServerSocket object.

See the following examples:

1. EchoServer.java
2. EchoClient.java

## A Multi-threaded Server

The EchoServer is a single-client server. It cannot service multiple clients *simultaneously*.

To overcome this limitation, we make the server threaded and call it ThreadedEchoServer.java.

The client, EchoClient.java, does not require any changes.

# Fetching URLs in Java

The program QueryHTTP.java connected to an HTTP server of a specified host and output the server's response to System.out.

The correct way to communicate with an HTTP server is to use the class **URL**. It has four constructors, two of which are

1. **URL**(String *protocol*, String *host*, int *port*, String *file*) throws MalformedURLException;
2. **URL**(String *protocol*, String *host*, String *file*) throws MalformedURLException;

The instance methods of the class include

1. URLConnection **openConnection**() throws IOException;
2. final Inputstream **openStream**() throws IOException;

The class **URLConnection** has a rich set of methods for gaining easy access to information used by HTTP such as

1. **getContentType**(); returns, for example, text/plain, text/html or application/octet-stream
2. **getContentLength**();

See page 336-337 of Java in a Nutshell for a complete description.

We use **openStream**() in the simple example FetchURLDemo.java to illustrate fetching a URL.

# Java and CGI Scripts

The interface with CGI in Java is primitive.

See CGIDemo.java as a simple example of a Java program that connects to an HTTP server and requests it to execute a perl script using HTTP GET.

Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/net.html

*java@nps.navy.mil*
Last Updated 26 AUG96
Naval Postgraduate School © 1996

# Threads

`java.lang` supports light-weight processes called **threads** which can be run in parallel.

Threads are run **asynchronously** within the same address space compared to heavy-weight processes which run in separate address spaces and traditionally communicate in UNIX environments using sockets and pipes.

With threads, interprocess communication and **context switching** are less expensive, providing the operating system supports threads.

Solaris 2.x and Windows95/NT support threads but **thread scheduling** actually varies among them; as yet, no Java semantics exists for thread scheduling behavior.

Threads can improve the performance of a system. For example, if a Java program needs to create a socket connection and create the widgets of a GUI, then the socket can be created using a thread so that if the connection is delayed, GUI construction does not block.

## The Thread Class

The Thread class has seven constructors. Among them are

1. **Thread**(Runnable *target*);
2. **Thread**(Runnable *target*, String *name*);

The Thread class (static) methods include

1. static Thread **currentThread**( ); returns current thread
2. static void **sleep**(long *milliseconds*) throws InterruptedException; makes the current thread suspend for a specified amount of time

See CurrentThreadDemo.java.

### The Runnable Interface

Each of the preceding Thread constructors takes an object of the **interface** Runnable:

```
public abstract interface Runnable {
        public abstract void run( );
}
```

Any class that implements Runnable (i.e. defines method run( )) can provide the body of a thread. See ThreadDemo.java.

### Daemon Threads

When does a concurrent Java program terminate?

When the only threads that remain are **daemon** threads. A daemon thread is one that exists only to provide a *service* to other threads.

A thread is asserted to be a daemon thread by the instance method:

final void **setDaemon**(boolean *on*) throws IllegalThreadStateException;

See Timeslicer.java.

## Thread Priorities

Every thread has a **priority**.

Thread priorities are integers in the range 1 to 10. They are used to determine which thread among a collection of enabled threads should be scheduled for execution.

A thread can **voluntarily** relinquish control by explicitly **yielding, sleeping, or blocking** on pending I/O. In this case, all threads are examined and the highest-priority thread that is ready to run resumes execution.

A thread can be **pre-empted** by a higher-priority thread that is ready to run.

When there are multiple threads of the same priority, they should EXPLICITLY yield control to their peers.

**Caution**: Currently JDK 1.0 for Windows95/NT will time slice in a round-robin fashion among threads of the same priority that do not relinquish control voluntarily. This is not so for Solaris 2.x. See the example HiLoPri.java.

Explicit time slicing is implemented by the class Timeslicer.java.

# Thread Synchronization

Why is synchronization important?

A *necessary condition* for the correctness of a concurrent program consisting of multiple threads is **serializability**.

Simply put, the outcome of a concurrent execution of some set of threads should be reproducible by a serial execution of them.

If the threads share a resource, then their concurrent execution may not be serializable.

Consider a shared bank account with two depositors. Both simultaneously deposit into the account. The

situation is simulated in BankDemo.java.

The program in Synch.java also demonstrates a concurrent execution that is not serializable.

## Synchronized Methods in Java

Java adopts the concept of a **monitor** introduced by Hoare. Monitors were used long ago in Brinch Hansen's concurrent Pascal.

But monitors are incorporated differently in Java.

A monitor can be thought of as a form of encapsulation with a *mutual-exclusion* property. The idea is that a process may call a method of the encapsulation after it has gained entry to the monitor.

At most one process can be "in" the monitor at any time. Following completion of the method, the process relinquishes the monitor to any other processes that may be waiting for it.

In Java, a monitor is an instance of a class with one or more methods of the class declared as **synchronized**.

Once a thread enters a synchronized method of an object, no other thread can enter any synchronized method of that object. Non-synchronized methods can still be called however.

For example, we can declare the **deposit** method in BankDemo.java to be synchronized in order to solve the two-depositor problem.

Another way to get serializability is to use the **synchronized statement**.

This is useful when you are using a class that was not designed for multi-thread access and thus has non-synchronized methods which manipulate internal state.

The synchronized statement has the form

**synchronized**(*object*) *statement*;

The statement is usually a call to a method of the object. The method call then will be handled as though the method had actually been declared **synchronized**.

For example in BankDemo.java, we can convert the run method to

```
public void run ( ) {
        synchronized (a) {
                a.deposit(amt, name, delay);
        }
}
```

# Inter-thread Communication

Why is thread communication in Java?

So far we have seen the need for mutually-exclusive access to a shared resource in order to satisfy **serializability**.

This is a **necessary** condition for the correctness of a concurrent (threaded) Java program, however, it may not be **sufficient**. Correctness may depend on other conditions as well.

Consider a *Producer* and *Consumer* as simulated in PC.java where Q1.java uses synchronized methods. Clearly the output of the program can be serialized.

Notice the need for synchronized **get** and **put** methods. If we use Q2.java instead, where the methods are no longer synchronized, then **get** can output n's value, **put** can then set n to a different value which **get** would then return.

Even though the output is serializable, it is unacceptable if there is an additional condition that the consumer must consume every value produced at most once. So suppose we take this additional condition as another correctness criterion.

What must we do then to make the program PC.java correct under this new criterion?

## Livelock

Consider modifying the queue of PC.java so that it uses **busy waiting** or **polling** as in Q3.java.

The problem with Q3.java is that either the Producer or Consumer may busily wait once inside the monitor which means that if it should be pre-empted, the other cannot call its queue method to break the busy wait. So the computation goes back into a busy-wait state.

Since the computation proceeds, but no progress will ever be made, the program **livelocks**.

The solution is to use inter-thread communication via Java's **wait( )**, **notify( )** or **notifyAll( )**.

All three methods can only be called from inside **synchronized** methods.

A thread can enter a synchronized method of an object and **wait** there until some other thread notifies it via **notify**. More precisely, we have

1. final void **wait( )** throws InterruptedException, IllegalMonitorStateException;
   forces the current thread to relinquish control of the monitor and to go to sleep until some other thread enters the same monitor and calls notify( ) or notifyAll( ).
2. final void **notify( )** throws IllegalMonitorStateException;
   wake up the first thread that called wait( ) on the same object.
3. final void **notifyAll( )** throws IllegalMonitorStateException;
   wakes up all threads that called wait( ) on the same object. The highest-priority thread awakened runs first.

*Reminder*. Waiting in a synchronized method relinquishes control of the monitor so that multiple threads

can enter the same method and wait for the same notification.

Finally, a solution to the Producer-Consumer problem is given in Q4.java.

As another example of the need for inter-thread communication, see

1.  OneCarBridgeTest.java; uses only synchronized methods to coordinate the use of a single-lane bridge assuming the maximum bridge load is one car.
2.  BridgeTest.java; uses wait( ) and notifyAll( ) to coordinate the use of a single-lane bridge assuming the maximum bridge load is three cars [Smith].

## Deadlock

A state where two threads have a circular dependency on a pair of synchronized objects.

See Deadlock.java.

Index


this page is located at
http://vislab-www.nps.navy.mil/~java/course/net.html

# Applet Security

The following material is based on information found in Exploring Java, Pat Niemeyer & Joshua Peck, O'Reilly.

The Java security model has three primary layers:

1. the programming language design layer
2. the byte-code verification/class loader layer
3. security manager layer

## The programming language design layer

This is often referred to as **implementation safety**.

Though not clearly defined, it is intended to capture protection against segmentation faults caused by dereferencing dangling or nonexistent pointers, or invalid address arithmetic.

It also is intended to capture protection against pointer forging as in C++

```
class CreditCard {
    private:
        char creditCardNo[16];
};
main( ) {
    CreditCard cc;
    // forge a pointer to peek inside class
    char *ccno = (char *)cc;
    printf("%s\n", ccno);
}
```

Java doesn't allow class types to be coerced, subverting encapsulation. Further, Java doesn't allow pointer arithmetic, does not have an "address of" operator, and has automatic memory management (garbage collection).

Another language-level issue is **final** classes which prevents class extension of classes that are part of the system.

## Byte-code verification and class loading layer

Classes that are loaded from over the network are subjected to **byte-code verification.**

Instructions of the Java virtual machine are typed. There are separate instructions for *stack operations* for object references and for each of the numeric types in Java. Further, there are separate *move* instructions for each type of value.

The type of an object resulting from an instruction is **unique**.

All this means that byte code can be *type checked* to ensure that instructions are used in ways they were intended to be used.

The verifier constructs a **type state** which consists of the types of class variables and types of values on the stack. It ensures that instruction types and stack value types match.

The verifier requires that all execution paths to the same point in the code arrive with the same type state, which means the same number of elements on the stack.

This rules out code that stacks values in a loop.

Therefore, if the code passes the verifier, we know what its stack requirements will be when executed. If the verifier finds that a requirement is *stack underflow* or *stack overflow* then it rejects the code.

# Security manager layer

The SecurityManager is responsible for making application-level security decisions.

A security manager is an object that can be installed by an application to restrict access to system resources.

It is consulted every time an application tries to access

1. file system
2. network ports
3. system properties

An instance of the SecurityManager class can be installed only once during an invocation of the Java interpreter. Thereafter, every access to a system resource is filtered through specific methods of the SecurityManager object by the core Java packages. Among these methods are

```
checkAccess(Thread g) --- Access this thread?
checkRead(String file) --- Read a file?
checkWrite(String file) --- Write a file?
checkDelete(String file) --- Delete a file?
checkConnect(Sring host, int port) --- Connect a socket to a host?
checkListen(int port) --- Create a ServerSocket?
checkPropertyAccess(String key) --- Access property associated with key?
checkTopLevelWindow(Object window) --- Create this top-level window?
```

All these methods simply return to grant access and throw a SecurityException otherwise except for checkTopLevelWindow( ), which returns true if access is granted and false if not; a value of false indicates the access is granted but that the new window should provide a warning border that it is untrusted.

See UntrustedWindow.java

The security manager installed by Netscape upon start up defines a security manager whose checkPropertyAccess( ) method throws a SecurityException when the key is one of the following properties:

1. "java.home" (Java installation directory)
2. "java.class.path" (the user's class path)
3. "user.name" (user's account name)
4. "user.home" (user's home directory)
5. "user.dir" (current working directory)

When the Java interpreter starts, it runs with a "null" security manager, that is one that grants all requests. That way the application can run like any other application with user privileges.

If the application needs more security, it can "install" an instance of the SecurityManager class using System.**setSecurityManager**( ).

See TinyHttpd.java and TinyHttpdSecurityManager.java.

Index


this page is located at
http://vislab-www.nps.navy.mil/~java/course/net.html

*java@nps.navy.mil*
Last Updated 11 SEP 96
Naval Postgraduate School © 1996

# CS2973 Final Project

- Student Portfolio

- Project Proposal Form

- Project Ideas

- Applet Template

- Project Schedule

---

Index
this page is located at
http://vislab-www.nps.navy.mil/~java/course/projects.html

*java@nps.navy.mil*
Last Updated 8 JUL96
Naval Postgraduate School © 1996

# CS2973 Student Portfolio

- O Star Student Projects Go Here!
- O

Index
this page is located at
http://vislab-www.nps.navy.mil/~java/course/portfolio.html

**java@nps.navy.mil**
Last Updated 8 JUL96
Naval Postgraduate School © 1996

# CS2973 Source Code Index

- Index of sample source code

- Images

- Handy Scripts for using the Root 222 lab remotely.

this page is located at
http://vislab-www.nps.navy.mil/~java/course/sourcecode/

*java@nps.navy.mil*
Last Updated 16 JUL96
Naval Postgraduate School © 1996

Index

# Student Connections

- Student Data Form

- Student Anonymous Feedback Form

- Homework Submission Form

- Student Pages and Email

- Java Documents

this page is located at
http://vislab-www.nps.navy.mil/~java/course/student_info/index.html

*java@nps.navy.mil*
Last Updated 21 JUL96
Naval Postgraduate School © 1996

# CS2973 Student Data Form

---

This form is **ONLY** for the students enrolled in or auditing the CS2973 course.
Please submit comments about the course or this website via the Feedback Form.

---

**Name:** *(Last,First, MI)* (required)

**Email Address:** (required)

**NPS login:** (if different than above)

**How did you hear about this course?**

**If "Other" please describe below:**

**What Operating Systems are you familiar with:**
Mac  Unix  Windows 3.1  Windows 95  Windows NT  Other  None

**If "Other" please describe below:**

**What programming languages have you used?**

**Remarks about your past assignments or experiences:**

Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/student_info/studentform.html
**java@nps.navy.mil**
Last Updated 8 JUL96
Naval Postgraduate School © 1996

# CS2973 Student Feedback Form

---

If your browser does not support forms, you may send email to
*java@nps.navy.mil*

---

Please provide feedback or comments about CS2973:

this page is located at
http://vislab-www.nps.navy.mil/~java/course/student_info/annoneval.html
**java@nps.navy.mil**
Last Updated 8 JUL96
Naval Postgraduate School © 1996

# CS2973 Student Homework SubmissionForm

Usage: a URL must be submitted for each exercise assigned in class. The URL must be submitted prior to the due time in order to receive full credit.

What is your email address? (the one used for class)

What is the complete URL to your assignment:
(ex: http://www.my.server/~mylogin)

this page is located at
http://vislab-www.nps.navy.mil/~java/course/student_info/hwsubmit.html
**java@nps.navy.mil**
Last Updated 21 JUL96
Naval Postgraduate School © 1996

# CS2973 Students

fnaltmisEmail Faik Nadir Altmisdort Homepage

ksbloodEmail Kimberely Blood Homepage

acamliguEmail Altay Camliguney Homepage

hacarveyEmail Harlan Carvey Homepage

cookEmail Mike Cook Homepage

erdoganEmail Ridvan Erdogan Homepage

jggarciaEmail Joe Garcia Homepage

dghuffEmail David Huff Homepage

calahtiEmail Carl Lahti Homepage

mpmccartEmail Mike McCarthy Homepage

rasmusstEmail Thor Rasmussen Homepage

evravidEmail Earl Ravid Homepage

msariEmail Mehmet Sari Homepage

pjszczepEmail Pete Szczepankiewicz Homepage

tjwerreEmail Timothy Werre Homepage

bdwhitakEmail Dane Whitaker Homepage

Email All CS2973

this page is located at
http://vislab-www.nps.navy.mil/~java/course/student_info/students.html

*java@nps.navy.mil*
Last Updated 23 JUL96

# Java links

## CS2973

links for the Java course

The Java Handbook: Examples
Java in a Nutshell Code Examples
Handy Software
History of the Internet
Hobbes' Internet Timeline
Object-Oriented Programming Concepts: A Primer

## java knowledge

JavaWorld - IDG's magazine for the Java community
Strong Java Mailing List WWW Gateway
java FAQ and tutorial
Ryan's HotJava HowTo!
The Java Developer
Java in India
Some Questions and Answers about using Java in Computer Science Curricula
Gordon H. Bradley home page
Teach Yourself Java: Errata
Java Land
The Java(tm) Page
Hostile Applets Home Page
Java Programming

## java sources

Dgclock in Java
Dr. Smith's Java(tm) Corner

## java links

TeamJava JAVA Links
Java Resources
Stratum Technologies
Diva for Java
JAVA IDE Comparison

## java tools

Free Tools for Java(tm)
Jlpr, the PostScript Applet

Iwan van Rienen's homepage
Java Disassembler
README.rtfd

## java tutorials

API User's Guide
Animation in Java Applets
Converting Objects to Strings
GUI Programming using Java
Index of all Fields and Methods
Java Packages
Methods for Adding UI Components
Packages
SunWorld Online - January - Java Developer
The AWT Tutorial
Creating a User Interface
The Java Package Tutorial
Trail Map: The Java Language Tutorial

## java - cool applets

http://www.nyx.net/~jbuzbee/mail.html
ChessLive!

## java script

Intellectual Technologies, Incorporated
Websys JavaScript Menu
JavaScript Index

## web stuff

**Instant NPS Webpage**
The WWW Security FAQ

## CGI/PERL

perl Programming
Smiley's CGI/PERL Source Page
perl
Matt's Script Archive
HTML Hacking Scripts
Dale Bewley's Perl Scripts and Links
The Scripts Home
Perl Tutorial

# APPENDIX B - LAB/HOMEWORK ASSIGNMENTS

The following Appendix provides copies of the CS2973 course Lab exercises and

Homework exercises as they appear on the Web pages. This Lab and Homework

Assignments were created in HTML format and are located at the web page address of

http://vislab-www.nps.navy.mil/~java.

Index

# CS2973 Homework Info

- Required WWW Setup
- Assignments
    - Homework 1
    - Homework 2
    - Homework 3
    - Homework 4
    - Homework 5

The CS2973 homework exercises constitute 60% of your final grade.
**Critical items are in bold.** Please adhere closely to these instructions so that all of the items are standardized.

## Required WWW Setup

If you do not have one, **create a home page** for yourself on the the NPS Web server. For more information look at the VisLab's HowTo or try a Page Generator to get up and running quickly. Ensure that your correct **email** address is listed on the page.

---

Top
Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw.html

*java@nps.navy.mil*
Last Updated 8 JUL96
Naval Postgraduate School © 1996

# Homework 1

# applet name

Student Name
date due

---

This applet uses the system clock to determine the time and uses the AWT to update with the paint method.

---

Win_Clock.java
Documentation


this page is located at
http://vislab-www.nps.navy.mil/~username/cs2973/hw1/hw1.html

*username@nps.navy.mil*

# Final Project Schedule

- Monday 16 September
    - Harlan Carvey
    - Mehmet Sari
    - Tim Werre
    - Mike Cook
- Tuesday 17 September
    - Mike McCarthy
    - Altay Camliguney/Faik Nadir Altmisdort
    - Dane Whitaker
    - Carl Lahti
- Wednesday 18 September
    - Ridvan Erdogan
    - Earl Ravid et al.
    - Kim Blood/Joe Garcia

this page is located at
http://vislab-www.nps.navy.mil/~java/course/project_schedule.html

*~java@nps.navy.mil*

# CS2973 Homework Exercise 1 (10 points)

Due 22 JUL 96, 1300

Create a home page for yourself on the the NPS Web server that has in it a local and nonlocal Applet running. Your page should look as follows:

## Your Name

*Local line Applet running here*

- Curriculum
- Email Address

*Nonlocal Applet of your choice running here*

The layout of the page must be as above. You must create the line Applet using `Rotator.java` and the image file `Line.gif`. Rotator.java is capable of rotating multiple images within a single file and it doesn't care what the images look like; it simply cycles through them at some specified rate of frames per second. In our case the frames come from `Line.gif`. The Java Source code can be found in `sourcecode` and the image file in `images`, each subdirectories of `~java/public_html/course`

The line Applet has three parameters. See the source code where they are documented. In addition to setting the width parameter to 500 pixels and the height parameter to 10 pixels, also set the frame rate to 30 and the background color to be the background color of your home page.

The nonlocal Applet must come from outside NPS and be referenced using a URL and the `codebase` tag of an Applet. See Section 15 of Java in a Nutshell for a description of this tag. Do not download your own personal copy of the relevant class file. For each Applet, you should provide some text that gets displayed instead by those Web browsers, like NCSA Mosaic, that are not Java powered.

When completed, the applets should reside in a web page titled "hw1.html" on any world readable server, though NPS is preferred. Email the URL of the page to *java@nps.navy.mil.*

this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw.html

*java@nps.navy.mil*
Last Updated 25 JUL96
Naval Postgraduate School © 1996

# CS2973 Homework Exercise 2 (15 points)

Due 5 AUG 1996, 1300

Create a standalone Java application `StrBufComp` such that

```
java StrBufComp arg1 arg2
```

outputs a negative number if arg1 < arg2, 0 if arg1 = arg2, and a positive number if arg1 > arg2, where < is lexicographic ordering with respect to the UNICODE character set. This is the same ordering one finds in a dictionary.

Define a class `StrBufComp` with a main method and a method called `CompWith` that takes two StringBuffer objects s1 and s2 and returns a negative number if s1 < s2, 0 if s1 = s2, and a positive number if s1 > s2. Method `compWith` should use method `charAt` which throws `StringIndexOutOfBoundsException`. You are not allowed to use the `String` method `compareTo`.

The main method must have an exception handler to catch the exception above and to catch `ArrayIndexOutOfBoundsException`. For the former exception, the handler should just output that the exception has been caught. The latter exception is thrown if an attempt is made to index into the command-line argument array at a position that does not contain a String object. (Such is the case if the user supplies less than two arguments.) So for the second kind of exception, the handler should output the following usage line:

```
usage: java StrBufComp arg1 arg2
```

There will be no need to prepare an HTML file for this exercise. After the due date specified above, I will look for the class file `~username/public_html/cs2973/hw2/StrBufComp.class` where `username` is your user name. I will run it and grade it accordingly, so it should be readable by all. I recommend that your source code for the class file also be readable by all, AFTER THE DUE DATE, of course. There is no need to email anything to user `java`. No email!

this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw.html

*java@nps.navy.mil*
Last Updated 25 JUL96
Naval Postgraduate School © 1996

# CS2973 Homework Exercise 3 (20 points)

Due 12 AUG 1996, 1300

This exercise is simple. You have to do only two things. First create a directory

~username/public_html/cs2973/hw3

that has only two files, one called BinaryTree.java and another called BinaryTree.class which is the class file created by compiling BinaryTree.java. Second, set your CLASSPATH environment variable by entering the following at the command line:

setenv CLASSPATH ".:/h/joshua_u2/java/PUBLIC"

Now here is how I intend to grade you. I will go to the directory you created in step (1) above, with my CLASSPATH variable set as above, and then run the appletviewer on ~java/PUBLIC/DictBox.html. This is a dictionary box Applet that allows me to create a table of words by entering a word at a time and pushing the **insert** button. It also allows me to **lookup** words that I previously entered in this way. If you're interested in seeing how it works, then go to the ~java/PUBLIC directory and fire it up using the appletviewer on DictBox.html. When I run the Applet from within your directory, namely ~username/public_html/cs2973/hw3, I expect to see the same behavior. That's why it is very important that your BinaryTree.java file be correct. When will you know it's correct? When it satisfies the specification below.

BinaryTree.java must implement two methods, one called **insert** and the other called **lookUp**. The former inserts a string into a table. The latter attempts to look for a given string in the table, returning the string itself if found and throwing SymbolNotFoundException otherwise. Their type signatures are given by the abstract class SymbolTable in ~java/PUBLIC/SymbolTable.java. You must extend this abstract class with a class called BinaryTree that implements these two methods using a **binary search tree** as the search data structure.

Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw.html

# CS2973 Homework Exercise 4 (15 points)

Due 19 AUG 1996, 1300

## Overview

This exercise is merely extending the `BinaryTree` class into a class called `Persistent.class` and adding the **Load** and **Save** functionality to the `Persistent.class`. The `DictBox.java` already provides the applet with four buttons: Insert, LookUp, Load, and SaveAs, and from HW3 the Insert and LookUp functionality should be there...your mission is to now allow a user to Load strings from the **Dictionary** file into the Binary Tree structure and then save the Binary Tree structure back to the file named **Dictionary**. If you're interested in seeing how it works, then go to the ~java/PUBLIC directory and fire i up using the appletviewer on DictBox.html. When I run the Applet from within your directory, namely ~username/public_html/cs2973/hw4, I expect to see the same behavior. That's why it is very important that your Persistent.java file be correct.

## Tasks:

1. First create a directory with the proper **PERMISSIONS** named:

**~username/public_html/cs2973/hw4**

that has only three (3) files:

- Persistent.java - this is the file that extends the BinaryTree.class (ie class Persistent extends BinaryTree { )
- Persistent.class - which is the class file created by compiling Persistent.java
- Dictionary - file that contains test data that you will add to...please leave the strings first, second, third, and fourth in the Dictionary file for grading. (Do not rename the Dictionary File, it is hard coded in the DictBox.java file, filename = "Dictionary"...and finally please makesure the file PERMISSION on this file is r and w ie chmod 766 Dictionary...this will allow me to test it properly.)

2. Set your CLASSPATH environment variable by entering the following at the command line:

**setenv CLASSPATH ".:/h/joshua_u2/java/PUBLIC"**

3. Add the Load and SaveAs Functionality:

- **Persistent.class** - The new DictBox.class applet has 2 new buttons added to it. Note how they work but you are not responsible for this file only the `Persistent.java` file. In this class you will extend the `BinaryTree.class` and add the file `load` and `saveAs` methods. Note the calls from `DictBox.java`
- **saveAs method of the Persistent.class** - This saves the Binary Tree to the Dictionary file. This

method will look like this:

```
public void saveAs(String dest) throws IOException {

}
```

Note that the saveAs method throws an IOException.
- **load method of the Persistent.class** - This method will look like this:

```
public boolean load(String filename) {

}
```

This method loads the Binary Tree with the strings into the Dictionary file.

## Standards:

Now here is how I intend to grade you. I will go to the directory you created in step (1) above, with my CLASSPATH variable set as above, and then run the appletviewer on ~java/PUBLIC/DictBox.html. When I test your code I will conduct the following tests:

- Insert string into structure via Insert Button.
- LookUp string in structure via LookUpButton - I will use the LookUp of first and second strings
- Load the Dictionary File via the Load Button. (This will allow a merge of the Binary Tree in memory with the one read in from the Dictionary File)
    - ○ Insert string then Load from Dictionary file: the structure should contain all strings
    - ○ Load from Dictionary file, then Insert string: the structure should contain all strings.
- I should get a not found response when I attempt to load the Dictionary file and it is not present.
- SaveAs to the file should put the elements in preorder.
- The following files should be in your directory: Presistent.java, Presistent.class, and Dictionary.
- The Dictionary file should have the new elements added to it when a SaveAs is done.
- Comments IAW Java Style Guide.


Index


this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw4.html

*java@nps.navy.mil*
Last Updated 11 Aug 96
Naval Postgraduate School © 1996

# CS2973 Homework Exercise 5 (25 points)

Due 23 SEP 1996, 1300

Rewrite the CS2973 Student Data HTML Form as a standalone Java form. You are required to use the GridBagLayout class. Components of the form should not grow with window resizing. Replace the "Send Info!!" button with a "Save form" button which writes the contents of the form to a local file in HTML with the same format as the form. The file to which the HTML is written should be obtained through use of class FileDialog.

You will note that the form has an option menu, editable text areas and checkboxes which are not radio buttons. You must preserve the behavior of the HTML form in your Java form. We recommend that you use Example 5-4 of Java in a Nutshell as a template for completing this exercise. Also see, in the same book, pages 112-114 and 255-257 for descriptions of the GridBagLayout and GridBagConstraints classes.

We recommend that you define a class called StudentFormComponents that loosely corresponds to AllComponents on pg 109 of the Nutshell. Add the form widgets to the frame and give a main method with no event handler, which will allow you to focus initially on just the form layout. Then extend this class with one called StudentFormEvents which does ALL event handling and has its own main method. You should not have to modify StudentFormComponents whatsoever. Create a directory ~username/public_html/cs2973/hw5 containing the files StudentFormComponents.java and StudentFormEvents.java and their byte code as well. And again, as a reminder, please give everyone read permission to these files at the time the assignment is due.

When viewed by Netscape, the HTML you generate should look like this. So take a look at the HTML for this using "view document source", to see what you should generate. Remember you must inspect the state of your checkboxes and output "checked" as part of the HTML tag **input** so that Netscape will display a checked box.

Index

this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw4.html

# CS2973 Project Proposal

This form is used to submit the project proposal.

**Name:** *(Last, MI, First)* (required)

**Email Address:** (required)

**Project Title:**

**Description of Project:**

---

this page is located at
http://vislab-www.nps.navy.mil/~java/course/proposal.html

*java@nps.navy.mil*
Last Updated 8 JUL96
Naval Postgraduate School © 1996

# Final Project Suggestions

1. The rudiments of a structured editor for Java. A structured editor allows one to edit parse trees rather than strings and to build a program by filling in placeholders from choices made available in a menu.
2. Extend the dictionary Applet to graphically display binary trees. The Applet might create a Frame or ScrollBar to display binary trees with at most 16 nodes. This would be a nice tool to illustrate how such trees can become unbalanced if strings are input in nearly sorted order.
3. Design a pocket Simon Applet. A pocket Simon has one audio tone associated with each of four different colors. The game begins with Simon playing one randomly-chosen tone and flashing its corresponding color. The user must respond by pressing that color. Now Simon replays the sound and randomly chooses another tone so that the user must now replay the two-tone sequence. Each time the user correctly replays the entire sequence, Simon adds yet another randomly-chosen tone to it for the user to replay. The game ends when the user can no longer replay the entire sequence at which time Simon may choose to offend you. Remember, Simon can vary the replay time so that the user hears the entire sequence more quickly. Oh boy!
4. Implement a client-server application; you might use as your starting point the chat client/server application described in the September issue of Dr. Dobb's Journal.

All of the above projects constitute a multi-person effort. If by Aug 26, user java has not received a final project proposal from you then you will be assigned to work on one of the above projects, perhaps with a colleague.

this page is located at
http://vislab-www.nps.navy.mil/~java/course/hw.html

*java@nps.navy.mil*
Last Updated 25 JUL96
Naval Postgraduate School © 1996

# APPENDIX C - RESOURCE GUIDE

The Resource Guide provides a guide for the Hardware and Software resources

required to implement a course using HTML and the Java programming language. The

Resource Guide provides a recommended set of hardware and software for the Macintosh,

Solaris, and Windows operating systems.

A.    Macintosh Operating System Version 7.5 or better:

    1.    Classroom

        (a)  Hardware

            (1) Mactionish Computer with:

                - 68 series processor or better

                - 36 MB RAM

                - 400 MB Free Harddrive Space

                - NIC or MODEM connected to local area network with

Internet Access

            (2) Overhead projection device that connects to a computer

            (3) Display screen

            (4) Power cord and surge protector

        (b) Software

            (1)  Mac O/S 7.5 or better

            (2)  Fetch 3.1 or better for File Transferring

            (3)  Sun Java Development Kit (JDK) version 1.1

            (4)  NCSA Telnet 2.6 or better for Telnet application

(5) Email application such as EUDORA

(6) Web Browser that supports Java such as Netscape 3.0

(c) Miscellaneous

(1) Pointer for screen

(2) Audio speakers for Computer

(3) Spare projector light bulb

2. Lab Area

(a) Hardware

(1) One Macintosh Computer per student and one for instructor

with:

- 68 series processor or better

- 36 MB RAM

- 400 MB Free Harddrive Space

- NIC or MODEM connected to local area network with

Internet Access

(2) Overhead projection device that connects to computer

(3) Display screen

(4) Power cord and surge protector

(b) Software (per machine)

(1) Mac O/S 7.5 or better

(2) Fetch 3.1 or better for File Transferring

(3) Sun Java Development Kit (JDK) version 1.1

(4) NCSA Telnet 2.6 or better for Telnet application

(5) Email application such as EUDORA

(6) Web Browser that supports Java such as Netscape 3.0

(c) Miscellaneous

(1) Pointer for screen

(2) Audio speakers for Computer

(3) Spare projector light bulb

B.     Solaris Operating System Version 2.3 or better:

1.     Classroom

(a) Hardware

(1) One Solaris Workstation for instructor with:

- 68 series processor or better

- CDROM

- 36 MB RAM

- 400 MB Free Harddrive Space

- NIC or MODEM connected to local area network with

Internet Access

(2) Overhead projection device that connects to a computer

(3) Display screen

(4) Power cord and surge protector

(b) Software (per machine)

(1) Solaris O/S 2.3 or better

(2) FTP Tool 3.1 or better for File Transfering

(3) Sun Java Development Kit (JDK) version 1.1

(4) Email application such as Open Mail

(5) Web Browser that supports Java such as Netscape 3.0

(c) Miscellaneous

(1) Pointer for screen

(2) Audio speakers for Computer

(3) Spare projector light bulb

2. Lab Area

(a) Hardware

(1) One Solaris Workstation per student and for instructor with:

- 68 series processor or better

- 36 MB RAM

- 400 MB Free Harddrive Space

- NIC or MODEM connected to local area network with

Internet Access

(2) Overhead projection device that connects to a computer

(3) Display screen

(4) Power cord and surge protector

(b) Software (per machine)

(1) Solaris O/S 2.3 or better

(2) FTP Tool 3.1 or better for File Transferring

(3) Sun Java Development Kit (JDK) version 1.1

(4) Email application such as Open Mail

(5) Web Browser that supports Java such as Netscape 3.0

(c) Miscellaneous

(1) Pointer for screen

(2) Audio speakers for Computer

(3) Spare projector light bulb

C.    Windows 95/NT Operating:

1.    Classroom

(a) Hardware

(1) Personnel Computer with:

- Pentium series processor or better

- CDROM

- 36 MB RAM

- 400 MB Free Harddrive Space

- NIC or MODEM connected to local area network with

Internet Access

(2) Overhead projection device that connects to a computer

(3) Display screen

(4) Power cord and surge protector

(b) Software

(1) Windows 95/NT O/S

(2) WS FTP PRO or better for File Transferring

(3) Sun Java Development Kit (JDK) version 1.1

(4) Telnet or better for Telnet application

(5) Email application such as EUDORA

(6) Web Browser that supports Java such as Netscape 3.0

(c) Miscellaneous

(1) Pointer for screen

(2) Audio speakers for Computer

(3) Spare projector light bulb

2.    Lab Area

(a) Hardware

(1) One Personnel Computer per student and instructor with:

- Pentium series processor or better

- CDROM

- 36 MB RAM

- 400 MB Free Harddrive Space

- NIC or MODEM connected to local area network with

Internet Access

(2) Overhead projection device that connects to a computer

(3) Display screen

(4) Power cord and surge protector

(b) Software

    (1)  Windows 95/NT O/S

    (2)  WINFTP  2.0 or better for File Transferring

    (3)  Sun Java Development Kit (JDK) version 1.1

    (4)  Telnet or better for Telnet application

    (5)  Email application such as EUDORA

    (6)  Web Browser that supports Java such as Netscape 3.0

(c) Miscellaneous

    (1)  Pointer for screen

    (2)  Audio speakers for Computer

    (3)  Spare projector light bulb

# APPENDIX D - A LOOK AT HTML

The World Wide Web (WWW) allows both information dissemination and information collecting through the capability of the Hypertext Markup Language (HTML). HTML is the programming language that has been standardized to be interrupted by WWW browsers (ie Mosaic, Netscape, or Microsoft Internet Explore). The current HTML standard is HTML 3.0. The HTML resides on a Hypertext Transport Protocol (HTTP) server which when invoked downloads the HTML to the requested client where it is interrupted by the clients' WWW browser. The clients' WWW browser then displays the Web page, the HTML downloaded, in the WWW browser in a WYSIWYG (what you see is what you get) format.

Representing traditional slide-oriented lectures in HTML requires learning the mechanics of the HTML and a set of software tools for Internet implementation. First we will look at the HTML Mechanics followed by the recommended tools.

## A.    HTML MECHANICS

HTML is the language used to create the documents for the World Wide Web. Although most browsers can display any document written in plain text, there are advantages to writing documents using HTML. When HTML documents are read by applications specifically designed for the Web, *i.e.* browsers or Web browsers, they can include formatting, graphics, and hyperlinks to other documents. In Figure D.1 below, the Netscape browser is displaying the HTML page for the CS 2973 course. Note that the underlined bullets are actual hyperlinks that can be clicked on to display other HTML pages.
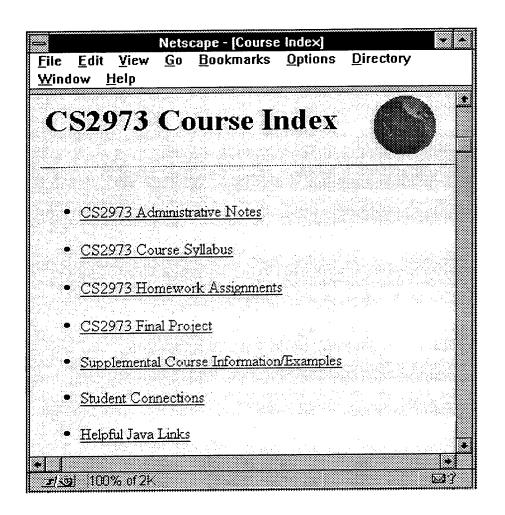
**Figure D.1. Example of HTML page viewed through the Netscape Browser.**

Two other ways to format text are Rich Text Format (RTF) and Postscript (PS) which concentrate on the appearance of a document. Unlike RTF and PS formatted documents, HTML concentrates on the structure of a document instead of emphasis on typefaces and illustrations. HTML emphasizes the need for basic, working components. For example HTML marks the headings, normal paragraphs, lists, numbering, and addressing. Additionally, HTML allows you to dynamically update the Web pages, which can be displayed over the Internet, with the use of a World Wide Web (WWW) browser.

HTML is a simple implementation of Standard Generalized Markup Language (SGML). HTML is a fairly simple language to use, however, there are editors to generate

it. HTML editors allow users to format documents in HTML format without having to learn the HTML scripting language. HTML editors can be useful, especially if writing massive quantities, but they are not necessary.

In general, HTML commands begin with a `<` and end with a `>`. The commands, not usually case sensitive, are "container" or "separator" commands. However, there are numerous exceptions. A container consists of a beginning command and an ending command. The commands are applied to the text between the beginning and ending commands. A container command is the title command, surrounding the text designated as the document's title. For example: `<title>` and `</title>`. A separator command is the command used to insert a linebreak `<br>`.

Non printable characters or "white space" is generally ignored in HTML. Leaving a blank line in your document will not create a blank line when the document is displayed in a browser unless you use the "preformatted" HTML tag. (*i.e.* `<pre>` and `</pre>`).

Not every element common to typical documents is included in HTML. Occasionally, there are problems converting documents. The commonly-used version of HTML does not support equations and support for tables is still relatively new. The implementation of many features also varies among browsers. For example, tables may look quite different from browser to browser.

There is a paradigm shift involved using HTML, especially for users used to controlling the look of a page. On the Web, the user and reader have more control over the look of documents than most publishers.

Some publishers attempt to circumvent the formatting limitations in HTML by using an abundance of graphics or non-standard HTML. However, using these techniques to make documents look "just right" in one particular browser, such as Netscape or Microsoft Explorer, often results in documents that cannot be displayed properly in other browsers.

It is far better to use general HTML commands than complex, non-standard commands because users see uniform documents regardless of the browser they use.

A HTML page consists of the following structures:

1.     **Heading.**

Every HTML document should start with a heading. The `<html>` command should be the first text inserted at the top of every document. The last text in a document should be the ending command, `</html>`. Headings are structured in levels.

a. Heading levels:

HTML is best suited for documents with a fairly rigid structure with definite outlines for headings, subheadings and lists. Although not required, it is good practice to write documents with heading "levels" which reflect the document's organization. For example: The first heading is a "level 1" heading, subheadings should be "level 2" and so on. Most browsers recognize at least four heading levels. There is support in HTML for more than four, but the browser only gives distinct styling on four levels. After the fourth level, it is difficult to distinguish between the heading levels. Beyond four levels, the best solution is to convert a single page document into multiple pages.

NOTE: The heading commands look like `<hX>` and `</hX>`, where X is the

heading level (1- 6). In most documents on the Web, the first heading is a

duplicate of the document's title.

b. Head.

The top of the document should have a section for heading information surrounded

by the `<head>` and `</head>` commands (Figure D.2 and D.3). Several items of

information should be the header, even though most browsers ignore this information. A

title must always be in the heading title. The title is surrounded by the `<title>` and

`</title>` commands. The title of a document is not normally displayed as part of the

page, but is displayed in a special section in most browsers. For example: Mosaic puts

the title in a Document Title box under the menu. Netscape displays the title on the Title

Bar. The title should be both descriptive and short enough to fit on one line

[HTML/CGI95].

```
<html>
    <head>
        <title>Course Index</title>
    </head>
    <body>
        <h1>Hello World</h1>
    </body>
</html>
```

**Figure D.2. Example of HTML with the first heading.**
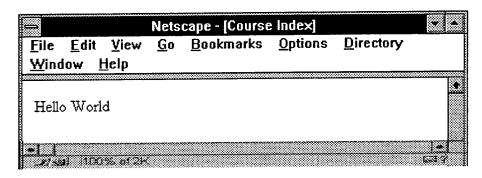
**Figure D.3. Example of HTML in the browser.**

## 2. Body.

The "body" of a document is everything after the head and should be marked with the

`<body>` and `</body>` commands (Figure D.4 and D.5).

```
<html>
    <head>
        <title>Course Index</title>
    </head>
    <body>
        HTML text of web page goes here
    </body>
</html>
```

**Figure D.4. Example of HTML structure with body command.**

Note: Remember white space is not directly interpreted, so all commands

can be on one line when necessary. Additionally, the body can consist of

numerous HTML commands to give the document the desired look.

164
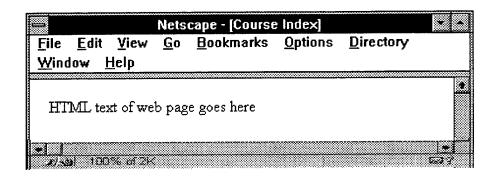
**Figure D.5. Example of HTML in the browser.**

a. Paragraphs.

Normal paragraphs are separated with the <p> command. This is an useful

command not normally used as a container, although it can be used if necessary. In

HTML, version 3.0, the <p> command is designed as a container so all paragraphs begin

with a <p> and end with a </p> (Figure D.6 and D.7). Most documents on the

web simply use the <p> command as a separator, but most browsers display paragraphs

correctly whether you use <p> as a separator or as a <p>container</p>. Using the

container form of the paragraph separator is the recommended way to formal paragraphs.

```
<html>
    <head>
        <title>Course Index</title>
    </head>
    <body>
        <h1>The title goes here.</h1>
        <p> This is a sample paragraph.</p>
        <h2>This is a subheading</h2>
        <p>HTML works great in education.</p>
    </body>
</html>
```

**Figure D.6. Example of HTML with paragraphs and a subheading.**
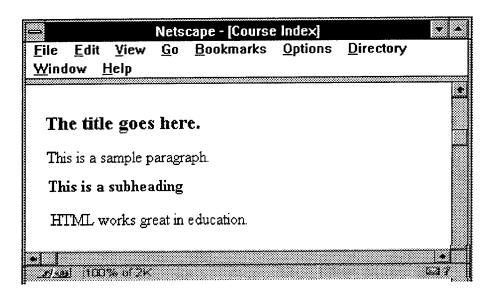
**Figure D.7. Example of HTML in the browser.**

b. Lists.

There are three kinds of lists in HTML: ordered, unordered, and a special

definition list (Figure 8 and 9).

(1) Ordered lists. Browsers insert list element numbers. This is convenient for

authors because inserted or deleted items in a sorted list are automatically updated to

reflect the correct numbering order. An ordered list begins with `<ol>` and ends with

`</ol>`.

(2) Unordered lists. Bullets mark each item in the list. The kind of bullet icon

displayed is up to the browser. For example: DOS may use asterixes or dashes. MAC

may use dots or circles. An unordered list begins with `<ul>` and ends with `</ul>`.

> NOTE: In both ordered and unordered lists, the individual items are
>
> designated with a `<li>` command. It is not necessary to separate items in
>
> a list with `<p>` commands. Additionally, you can also nest lists to get an
>
> outline effect.

```
<html>
   <head>
      <title>Course Index</title>
   </head>
   <body>
      <h1>The title goes here</h1>

      This is a sample paragraph.

      The majority of most documents contain

      this type of construct.

      <p>

      <h2>This is a subheading</h2>

      <p>HTML is good for education.

      ordered List:<p>

      <ol>

         <li> first item.
         <li> second item.
         <li> third item
      </ol>

      <p>unordered list:</p>
      <ul>
         <li> an item.

         <li> another item.

         <li> here's a nested list

      <ul>

         <li> a nested item

         <li> another nested item

      </ul>

      <li> the last item

   </ul>
  </body>
</html>
```

**Figure D.8. Example with multiple lists.**
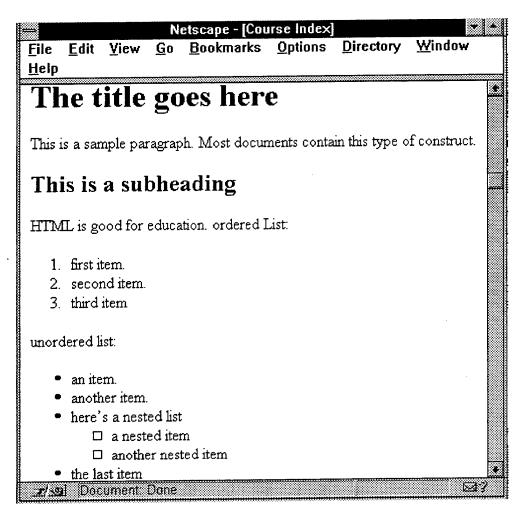
**Figure D.9. Example of the above list when viewed with the browser.**

(3)    Definition lists. A definition list is very flexible and is used only for lists

needing explanatory text for each item. Each item in the list has two parts - a term

indicated with the <dt> command and a definition which uses the <dd> command. The

list itself starts with a <dl> command and closes with a </dl> command (Figure 10 and

11) [HTML/CGI95].

```
<dl>
  <dt> First Term
     <dd> First term's definition.
  <dt> Second term (or title)
  <dd> Text that explains the second term.
</dl>
```

**Figure D.10. Example of a sample definition list.**
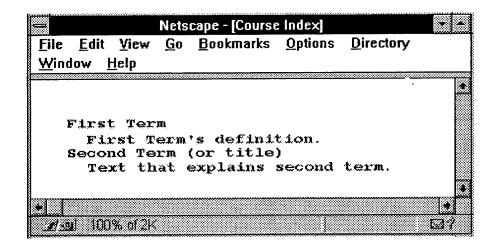


**Figure D.11. Example of what it would look like in the browser.**

## 3. Links.

Links are what make Web documents unique. Unfortunately, creating a link is slightly

complicated. The most complex part is the URL pointing to the resource link.

a. URL (Universal Resource Locator). The address of a document or resource.

`protocol://machine.name[:port]/directory/document.name`

The protocol is the transfer protocol used to reach the document or resource. On

the Web, it is typically "HTTP", but it can also be numerous other things - ftp, gopher,

telnet, etc. The machine.name is the name of the host where the document resides -

i.e. vislab-www.nps.navy.mil. The ":port" portion of the address is optional and

is only necessary when the protocol is listening to a non-standard TCP port number. The

standard port number for HTTP is 80. There are numerous Web servers on the Internet

using non-standard ports, such as port 8000. To reach a gopher at something than port 70

or telnet at something other than port 23, you have to put in a port number.

The easiest way to access a URL in a HTML document is to copy the URL into

the document via a web browser; Netscape, Mosaic, or Cello. For example: in Mosaic,

copy the text in the Document URL field near the top of Mosaic's window. In Cello, click

on the link with the right mouse button and choose the "copy" command. In Netscape

either copy the text in the "Location" box at the top of the Window or click on the link

with the right mouse button and choose" copy link to clipboard."

b. Putting Links in HTML documents. The HTML command for putting a link

into a document is: <a href="URL"> text of link </a> (Figure 12

and 13). Put the URL in the quotes following the "href=" and put the text of the

link, the part that users will click or select to activate the link, after the > and before the

</a> [HTML/CGI95].

```
<html>
  <head>
   <title>This is my title</title>
  </head>
  <body>
    <h1>The title goes here</h1>
    <p> This is a sample paragraph.
    <a href="http:vislab-www.nps.navy.mil">here
    </a>.</p>
    <h2>This is a subheading</h2>
    <p>HTML is good in education.
    <a href="http://vislab-
    www.nps.navy.mil/~java/planner.html">
    Java Stuff</a>
  </p>
  </body>
</html>
```

**Figure D.12. Example using links.**
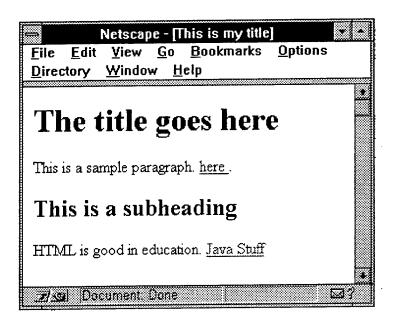
**Figure D.13. Example of what it would look like in the browser.**

4.      **Images.**

It is not difficult to put an image into a web page. One of the greatest advantages

of using the Web is creating and sharing documents across platforms. However, restraint

is key. Users frequently turn off in-line images to increase performance. Even with fast

network access to the Internet, some users are annoyed by documents loaded down with

images. However, a dash of colorful images maybe needed to make a point. Additionally,

in education the use of images are instrumental in conveying an idea..

In order to add an image you must use the `<IMG SRC="image.gif>`

command. Images added to a document need to be converted into Graphics Interface

Format (GIF) or Joint Photographic Experts Group (JPEG) format. There are a number of

tools currently in use. Industry standards include XV on the UNIX platform and

PaintShop Pro on the Windows/NT platform. These tools are valuable for converting and

editing images to be placed into web page documents. The location does not have to be a

full URL, but it can be. The same trick can be used for normal links, as well as images. There is not a noticeable difference in speed using relative URLs opposed to full URLs. It is really just a matter of user preference.

The HTML command for inserting an image at the current position is:

(1) **Relative URL**: `<IMG SRC="name_of_image.gif/jpeg">`

(2) **Full URL**: `<IMG SRC="vislab-`

    `www.nps.navy.mil/~java/name_of_image.gif/jpeg">`.

NOTE: It is possible to display a GIF/JPEG images stored almost anywhere, including on the Internet, but it is more complex. It is simpler to create a directory named `images` in which to store all images (ie. ~your_html_directory/images).

There is an optional way to use the IMG command. A "suggestion" can be made to the browser to align the image in a particular way with respect to the surrounding text. For example, use the `"align="` directive. The alignment choices are `"top"`, `"middle"`, or `"bottom"`, which indicate where the base, bottom edge, of the image should be in relation to the base line of the surrounding text (Figure D.14).

```
<img align="top" src="./image.gif"> Top Text.
<img align="middle" src"./image.gif"> Middle Text.
<img align="bottom" src"./image.gif"> Bottom Text.
```

**Figure D.14. Example of aligning items with commands.**

When viewed in a browser, the HTML code described in Figure D.14

would be displayed as in Figure D.15.
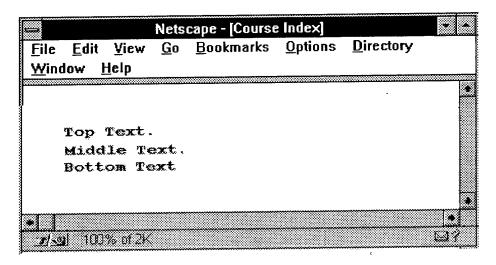


**Figure D.15. Example of how it looks in the browser.**

Another useful option is to use the "alt" directive to "suggest" a text-only

alternative for browsers that do not support in-line images.

```
<img alt="[sample image #1]" src="./image.gif"> Sample Text.

<img alt="[sample image #2]" src="./image.gif"> Sample Text.

<img alt="[sample image #3]" src="./image.gif"> Sample Text.
```

**Figure D.16. Example using "alt" tag.**

For users on a text-only browser, *i.e.* W3-mode of Emacs or Lynx, these items

appear "[sample image]" instead of [IMAGE] (Figure 16 and 17). Some web servers use

this directive to display icons for image-oriented and simple word links. For example

" [Home]  [Next] " for text-only browsers [HTML/CGI95].
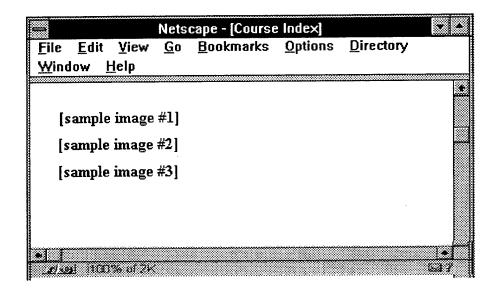
**Figure D.17. Example of HTML in the browser.**

5. Applet Tag.

The Applet Tag is a Java specific HTML tag. It is composed of several elements.

```
<APPLET
  CODEBASE = applet-url
  CODE = applet-filename
  WIDTH = pixel-width
  HEIGHT = pixel-height
  [ALT = alternate-text]
  [NAME = applet-name]
  [ALIGN = alignment]
  [VSPACE = vertical-pixel-space]
  [HSPACE = horizontal-pixel-space]
>
[<PARM NAME = parameter VALUE = value>]
[<PARM NAME = parameter VALUE = value>]
...
[altermate-html]
</APPLET>
```

**Figure D.18. Example of the Java HTML Tag semantics.**

The first element is a <applet> which indicates the start of the Applet Tag. The second element is the CODE ="HelloWorld.class", which indicates the file containing a compiled applet (bytecode) (Figure 18, 19 and 20 ). The third and fourth elements of the Applet Tag indicate the width and the height of the applet in pixels. The upper left corner of the applet is always at x-coordinate 0 and y-coordinate 0. The width of this applet is 275 pixels and the height of this applet is 35 pixels. Additionally, the Applet Tag can have numerous variable parameters that can be passed from the HTML page to the applet, although not required. The variable parameters are passed to the applet via the [<PARM NAME = parameter VALUE = value>]. The fifth element is the </applet> which signals termination of the applet tag [Nutshell96].

```
<APPLET
CODE     = "HelloWorld.class"
WIDTH   = "200",
HEIGHT = "200"
>
[<PARM NAME = "BGCOLOR" VALUE = "red">]
</APPLET>
```

**Figure D.19. Example using Java Applet in HTML Page.**
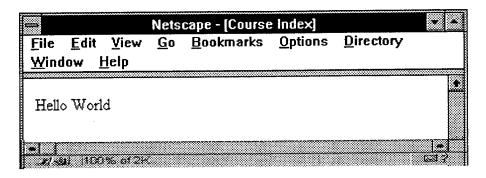
**Figure D.20. Example of HTML in the browser.**

## B. HTML SOFTWARE TOOLS

In this section we describe some of the tools that are useful in developing Web applications. The minimum tools necessary are: a text editor to create and edit the HTML documentation; an image editor supporting GIF and JPEG images which allow editing and resizing of images; a file transfer protocol (FTP) tool allowing the transfer of files between workstations and the HyperText Transfer Protocol (HTTP) server; and a web browser for viewing the created HTML documents.

**Recommended tools for developing a web page document:**

1. HTML authoring tools:

   a. WINDOWS - Homesite 2.0,

   *http://www.dexnet.com/index.html?/apps/homesite.html*

   b. MACINTOSH - Page Mill 2.0,

   *http://www.adobe.com/prodindex/pagemill/main.html*

   c. UNIX - HotMetal 3.0, *http://www.sq.com/products/hotmetal/hmp-org.htm*

2. Imaging Tools:

    a. WINDOWS - Paint Shop Pro 4.1, *http://www.jasc.com/psp.html*

    b. MACINTOSH - Photo Shop 4.0,

    *http://www.adobe.com/prodindex/photoshop/main.html*

    c. UNIX - XV, *http://www.sun.com/sunsoft/catlink/xv/xv.html*

3. FTP Tools.

    a. WINDOWS - WS FTP PRO, *http://www.ipswitch.com/pd_wsftp.html*

    b. MACINTOSH - Fetch 3.0.1, *ftp://dartmouth.edu/pub/mac/*

    c. UNIX - FTP Tool, *http://www.sun.com*

4. Browsers:

    a. WINDOWS - Netscape 3.0, *http://www.netscape.com*

    Microsoft Explorer, *http://www.microsoft.com*

    b. MACINTOSH - Netscape 3.0, *http://www.netscape.com*

    Microsoft Explorer, *http://www.microsoft.com*

    c. UNIX - Netscape 3.0, *http://www.netscape.com*

    Many of the HTML development tools we found to be very similar in performance and functionality. The basic recommendation is to find a set of tools you feel comfortable with and that best meets your needs.

## C.    SUMMARY

    The use of HTML to create the course lecture notes proved to be valuable in allowing students to view notes in the classroom, as well as on the Internet at anytime.

Additionally, the use of HTML allowed the course notes to be updated and distributed to

students immediately. To view the course lecture notes, see Appendix A, Lecture Notes,

or http://vislab-www.nps.navy.mil/~java.

# APPENDIX E - A LOOK AT THE JAVA AWT

This appendix provides an overview of the Java Abstract Windowing Toolkit (AWT). A comprehensive treatment of Java and its other API packages is outside the scope of this appendix. For additional information, the books "Core Java" [CoreJava96], "Exploring Java" [Exploring96] and "Java in a Nutshell" [Nutshell96] are recommended.

## A.  The Java Programming Language

There is no doubt Java has received much attention since its official announcement by Sun Microsystems in May 1995. Whether much of the excitement surrounding Java is really justified remains to be seen. However, it is clear it's a simple, yet flexible, alternative to C++ for many applications.

Java is an Object-Oriented Programming language originally designed for writing code to control devices such as VCR's, set-top boxes for interactive TV, and other assorted appliances. Its primary application today, Web programming, was largely unforeseen when it was originally designed in 1991 and called Oak. Now it is used to develop "Applets", relatively small programs downloaded from an HTTP server by a Java-enhanced Web browser (e.g. Netscape Navigator), and executed by the browser on the client's machine. The downloaded code comprises instructions of a virtual machine (the Java VM) and, therefore, can run on a variety of different platforms (e.g. Solaris and NT) as long as these platforms implement the Java VM. This kind of portability is one of the major reasons for the widespread appeal of Java.

Another major reason for Java's appeal is its support for bringing interactive content
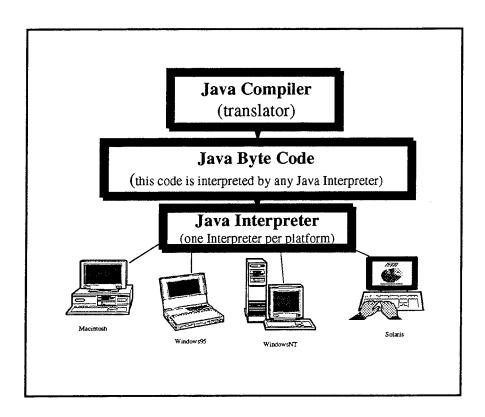
**Figure E.1    Java Flow Diagram**

to Web pages.  Prior to Java, Web page interaction was limited to static HTML forms and

CGI scripts.  The Java Application Programming Interface (API) provides a useful set of

methods for writing general graphic user interfaces (GUI's).  The API is a collection of

"packages", each containing "class" specifications.

## B.    The Java AWT

The Abstract Windowing Toolkit is a platform-independent windowing toolkit

supporting many of the basic needs of a GUI (e.g. windows, menu bars, buttons,etc).

The toolkit is regarded as abstract because it serves as an interface between Java

applications (and Applets) and native GUI libraries such as Motif.  Details of the

underlying GUI library, which are platform-specific, are hidden from Java programs

making the Java program more portable between operating systems. Current platforms the AWT has been implemented on include Microsoft Windows95/NT, Motif and Macintosh 7.5. Because the windowing subcomponents, buttons, scrollbars, etc., are actually provided by the native windowing system's "peer objects", there may be differences between platforms (Figure E.4). For instance, the file dialogue on the Macintosh is different from that of Microsoft Windows. In order for the Java AWT to be portable, it must have an interface developed for each specific platform allowing the AWT to recognize the native system's peer objects. In overall appearance, the AWT is similar in functionality to early versions of Motif. The Java AWT provides the "highest common factor" of functionality across a wide variety of native windowing toolkits, (see Figure E.1).

At first glance, the structure of the AWT is complex. The AWT consists of 23 classes, consisting of nine different components added to, and then laid out, by five different layout managers in two different types of containers (panel or window). The AWT is simply defined as a set of nested components, starting from the outermost component, i.e. window, all the way down to the smallest UI component, adhering to the "is a" inheritance model, (see Figure E.2). Additionally, there are a variety of event handling, menu, fonts and graphics classes used in supporting the AWT. However, the AWT alone is not adequate for complete GUI development. Threads, audio, I/O, and network classes should be implemented to produce full functioning Applets.

Now we will look at the java.awt.Component class and its subclasses consisting of:

- Subcomponents

- Containers

- Windows

- Panels and Layouts

## C.    The java.awt.Component Class

Components are the window icons with which the user interacts - Buttons,

TextAreas, Scrollbars, etc.  Basically, Components are the visible GUI controls added to a

Container.  Anything derived from the java.awt.Component class can be a Component.

Some common methods used in the java.awt.Component package are:

getBackground()

setBackground(Color)

getFont()

setFont(Font)

mouseDown(Event, int, int)

show()

resize(int, int)

paint(Graphics)

update(Graphics)

move(int, int)

## C.1 Subcomponents

Subcomponents and their constructors are added using the add(Component) or

addItem(String) method and removed with remove(). Adding a component doesn't
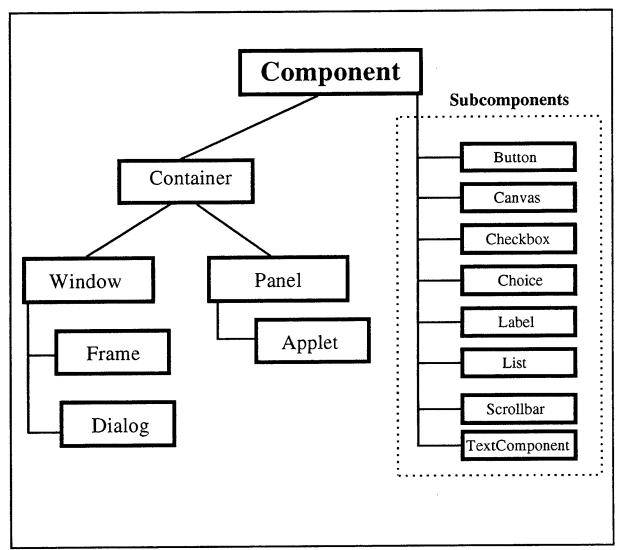


**Figure E.2   Structure of AWT (java.awt package)**

necessarily create the native Subcomponent (peer Subcomponent) for that component, but

simply inserts the component into the component hierarchy. The nativeSubcomponent is

only instantiated when pack() or show() is called on a window or when you add a

component to a container whose native subcomponent is already created. The Subcomponents provide basic controls, labels, and methods for creating custom components for all types of GUI interfaces. Many trigger ACTION_EVENTs for interactive applications.

The Button creates a simple on-screen button the user can push. A string within the constructor shows up as a label on the button.

Button();

Button(String);

The Canvas provides a semantic-free, nearly abstract class which inherits its functionality from the Component class. The Canvas is well suited for building custom components. It is ready for use as a drawing surface and allows the implementation of event handling. When the Canvas is extended, inherited methods are overwritten to provide the desired functionality.

The Checkbox has a selected/deselected toggle control and may also have a label. It may be initialized with either true or false boolean value. A series of checkboxes in the same CheckboxGroup will be mutually exclusive.

Checkbox();

Checkbox(String);

Checkbox(String, CheckboxGroup, Boolean);

The Choice creates a pop-up menu on the screen when the user clicks on it. Items are listed in the order added to the class with addItem. Only the first item added will be displayed on top.

Choice C = new Choice();

C.addItem("First choice");

The Label displays a static line of text, although it may be constructed empty and set at a later time. Labels also have an alignment property of LEFT, CENTER or RIGHT stored as an integer.

Label();

Label(String);

Label(String, int);

The List displays a scrolling list of items of which one or many may be selected. A List can specify the number of lines to be visible or use the size of the container as the default. A boolean is set to determine if multiple selections are permitted.

List itemlist = new List(1, true);

itemlist.addItem("first item");

itemlist.addItem("second item");

The Scrollbar displays the familiar windows control for moving through a range of values. The orientation is set to HORIZONTAL or VERTICAL. The initial value or position of the scrollbar index is set to an integer value. The visibility or overall area displayed is specified in pixels. The minimum and maximum size of the scrollable area are also specified in pixels as integers.

Scrollbar(orientation);

Scrollbar(orientation int, initial_value int, visible int, minimum int, maximum int);

The TextComponent superclass of TextArea and TextField supplies many methods which allow the programmer to control the behavior or place restrictions on a TextArea or TextField. For instance, the user can restrict the behavior of a TextArea or TextField such that the text within the TextArea or TextField is editable or not editable.

The TextField creates field with a single, editable line of text. The number of columns in the text field may be set with an integer, and the text initially displayed is set with a String. TextField also has the useful setEchoChar() method which is useful for password fields.

```
TextField();

TextField(int);

TextField(String);

TextField(String, int);
```

TextArea creates a multi-line box for text entry. The TextArea may be any height and width-scrollbars are added by default. The int arguments set the number of rows and columns (respectively) of the area. The appendText() method allows text to be added to the TextArea incrementally. Other methods include insertText(String, int) and replaceText(String, int, int). Like the TextField class, TextArea has four constructors.

```
TextArea();

TextArea(int, int);

TextArea(String);

TextArea(String, int, int);
```

## C.2 Containers

The Container class is an abstract subclass of Component and, is designed to group together related components. A Container may also contain other Components. The two subclasses are Windows, which provide the windows to contain Components, and Panels, which group Components within the area of an exiting window. In general, Components must be added to a Container before they can be used or modified. They use the add() method with one or two arguments depending on the LayoutManager used. Common methods used in Containers are:

- add(Component);
- add(String, Component);
- remove(Component);
- getComponents();
- setLayout(LayoutManager);
- getLayout();

## C.3 Windows

When an application is written for a GUI using the AWT, the Window class provides the framework for a basic standalone process. A standalone process is a Java application which executes within the operating systems API without the use of the browser's framework. The Window is the next major subclass of Container. It provides a top level window with no borders or menu bar. Although it is rarely called directly, it does provide the important event handles WINDOW_DESTROY, WINDOW_ICONIFY, WINDOW_DEICONIFY and WINDOW_MOVE. The two Window subclasses are Frame and Dialog.

The Frame subclass of Window provides a container for all associated components used in the main window of the program. The Frame has many predefined convenience methods and may have a specified title, menu bar, icon and cursor. The Frame does not need to be displayed physically within any other type of container or component, it can stand freely on its own. The show() and hide() methods make the Frame appear and disappear when needed. The setResizable(boolean) determines whether the user can resize the Frame. Frame is the only container class supporting the MenuComponents class and subclasses.

The MenuComponent class is extended from the Object class, so it is not a component that can be usable anywhere or instantiated directly. The subclasses using MenuComponent are MenuBar, Menu, MenuItem and CheckboxMenuItem. The MenuBar sets up the platform-dependent group of menus, typically across the top of the window. Each menu and the drop-down list of options is represented by a Menu object. Each object on the list is a MenuItem. Menu is implemented as a subclass of MenuItem, so sub-menus can be made by adding one menu to another. For example, CheckboxMenuItem is a subclass of MenuItem class.

The Dialog class provides a simple, encapsulated dialog box which blocks further user input until dismissed. Dialogs are dependent on other windows. If the parent window is iconified or destroyed, the Dialog box mirrors its behavior. A handy subclass of Dialog is the FileDialog which brings up the native file chooser window to allow local file selection. FileDialog currently has Load and Save buttons built in, but other operations can also be supported.

Dialog(Frame parent, boolean modal)  no title on window.

Dialog(Frame parent, String title, boolean modal)

FileDialog(Frame parent, String title)

## C.4 Panels and Layouts

A Panel is one of the most generic, yet concrete, containers available which can be

displayed on-screen.  Once a component is added to a Panel, it can be manipulated using

the move(), resize() or reshape() methods inherited from Component.  Each Panel or sub-

panel may be treated as a Container in its own right.  To offer more control and flexibility
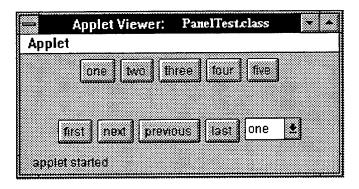


**Figure E.3   Sample PanelTest.class Applet  (see Appendix E-1)**

in how components are displayed within a container, the abstract classLayoutManager is

used.  The final appearance of a component is determined by which LayoutManager is

used, and the order in which items are added to the container.  Figure E.3 is a graphic

display of the use of the Panel class and the LayoutManagers, while Appendix E-1

provides the Java source code which created this Applet.

Layouts define how Components are "laid out" within a Container.  There are five

predefined subclasses of the LayoutManager.  Since the LayoutManager class is abstract,

it cannot be used directly.  Instead, it must be a sub-class with its own functionality or,

191

use a derived class defined in the AWT - i.e. BorderLayout, CardLayout, GridLayout, FlowLayout, etc. The actual appearance of the AWT components on-screen is determined by the order in which they are added to the Panel. (and the LayoutManager class the panel is currently using to lay out the screen.) The LayoutManager class determines how portions of the screen are sectioned and how components within the Panel are placed.

The BorderLayout arranges components around the edges of the container according to an area specified in the instantiation of the component. The areas of choice are: north, south, east, west, and center. The area in which components are added is important. If a center component is added it gets allocated all remaining space. The other areas expand only as much as necessary to keep all available space filled. The argument add() method is used to add a component with BorderLayout. If a gap is desired between components, the horizontal and vertical pixel values are set with the setLayout method.BorderLayout is the default layout manager for all Windows, such as Frames and Dialogs.

Add("direction", new Component("string"));

BorderLayout(int horizontalGap, int verticalGap);

The CardLayout gets its name because only one visible display space is visible although many Containers may use the same space. Like a stack of playing "cards", components may be displayed by calling the methods first(), last(), next(), previous(), or show() to get a specific card. Again, the argument add(String name, Component comp) is required with the String being an identifier for that component. The order in which components are added determines their order in the "deck".

Panel card1 = new Panel();

card1.setLayout(new CardLayout());

```
add("first", card1);

card1.show(card1,"first");
```

The GridLayout provides an easy method for organizing components in a grid. All components are placed in equally sized cells, and placed in the container in the order in which they are added, starting in the top row from left to right. If a window containing a GridLayout is resized, the cells grow or shrink so that all the space available to the container is used. The GridLayout constructor includes the number of rows and columns required. As an option, horizontal and vertical gaps maybe specified in pixels. At least one of the row arguments or column arguments must be non-zero.

```
GridLayout(int rows, int columns);

GridLayout(int rows, int columns, int horizontalGap, int verticalGap);
```

The GridBagLayout is much more flexible than GridLayout as it allows specific components to span multiple rows or columns. The characteristics of each component is optionally set with GridBagConstraints. Various options may be set including anchor position; grid position, which may be relative or absolute; the number of horizontal and vertical cells occupied; fill direction; margin insets; and directional weights.

```
GridBagLayout();
```

The FlowLayout is probably the simplest to use. It puts components in a row until the horizontal space is full, then shifts to multiple lines. Horizontal and vertical gaps may be specified with a default of five pixels. The alignment of the components on the rows may be LEFT, RIGHT, or CENTER justified when the FlowLayout is created. FlowLayout is the default layout manager for all Panels.

FlowLayout();

FlowLayout(int alignment);

FlowLayout(int alignment, int horizontalGap, int verticalGap);

## C.4.1.  Applet

An Applet is a subclass of a Panel designed to tie the Container to the browser of the applet viewer.  Applets inherit their paint() and update() methods from the Applet class, which hierarchically, inherit them from the AWT Component class (Figure E.2).  The repaint() method saves time and resources by only redrawing the part of the component that has changed.  Since Applets are Panels, they display the same features as all Panels. Because Applets can appear in a browser or viewer window, they do not need to build a Window.  Applets have the ability to use resources such as audio clips, images, or other data files on the internet since they are accessed via the Universal Resource Locator (URL).  For security reasons, restrictions may apply if the applet is loaded over the network.  An Applet loaded over a network may not:

- Define any system properties

- Read or write files on the local host

- Start  programs on the local host

- Load libraries or define native methods

- Make a network  connection to any computer other than the one from which the applet was itself loaded

Applets are capable of doing many functions other applications cannot.  Applets loaded locally are not restricted.  Applets can load other HTML documents in a browser, and may continue running after exiting the window which invoked the Applet.  Overall,

Applets are a very innovative and powerful way to send dynamic applications over the Internet, without having to worry about the native machines operating system environment.

GUIs and Windowing systems use event-driven models rather than procedural-grab-the-input-yourself models. The O/S polls constantly for events. When an event is located, it is relayed to the program and the program decides how to handle that event. An Event object is created when a user triggers it, by clicking a button or pressing a key on a Component. The Component deliverEvent() method takes the Event object from the top-most Component, such as a browser or a Window, and sends it down to the lowest Component, such as a Button. Once the target component is reached, the AWT allows each Component to handle the event at every step of the Component hierarchy. At each component level, the Event handler may modify the Eventinstance before it is passed up, stop the Event from being processed further, or react in some other way by doing interim processing. Each Event object includes:

- The type of the event (a key press or mouse click)

- The object that was the "target" of the event (the Button clicked)

- A timestamp indicating when the event occurred

- The location (x,y) where the event occurred

- The key that was pressed (for keyboard events)

- An arbitrary argument (such as the string displayed on the Component)

- The state of the modifier keys when the event occurred

Unhandled events are passed to the parent container via postEvent() so they get a chance to deal with the event. Components can respond to events by implementing a custom-written handleEvent() method or by using the default (Component) definition of handleEvent() which simply calls a method specific to the event type. The following events routinely need to be handled in a typical Java Applet

- ACTION_ EVENT - The ACTION_EVENT event occurs when you press a button, select a menu, etc

- MOUSE_DOWN - The MOUSE_DOWN event occurs when the mouse button is clicked down

- KEY_ACTION - The KEY_ACTION event occurs when a key on the keyboard is pressed

- WINDOW_DESTROY - The WINDOW_DESTROY event occurs when the window of the object is closed

## D. Peer Objects (Package java.awt.peer).

The java.awt.peer package is a subpackage of the AWT package providing interfaces implemented by the native windowing system. The java.awt.peer package does the work behind the scenes allowing the Java program to interface seamlessly with specific platform native peer objects. The native peer objects are the objects in the native windowing system allowing a button, scrollbar or other GUI interfaces to be produced in accordance with the native windowing system methods. The native peer objects are derived from the abstract java.awt.Component class, ( Figure E.2). There is a native peer object corresponding to each java.awt.Component class object. A button displayed by a Macintosh windowing system is different from a button displayed by Windows 95/NT

windowing system. The power behind the java.awt.peer package is that it is neutral. It simply makes a request to the native windowing system for a Button, with certain dimensions, and interfaces with the native windowing system to display the Button in the Java Applet.

The ability to make a request to various native windowing systems is handled through the java.awt.Toolkit package. The java.awt.Toolkit is the core of the Java VM which the AWT uses to talk with the native windowing systems. The java.awt.Toolkit is a package that creates objects in the native display system. The java.awt.Toolkit contains a number of methods for creating "peer objects", between the Java program and the specific native widowing systems. The java.awt.Toolkit is the one part of the Java paradigm "not" fully portable, but instead it is developed for specific platforms. The java.awt.Toolkit contains methods for making instances for each type of java.awt.peer component. As stated before, currently this portability only applies between Macintosh 7.5, Motif, and Windows95/NT. However, Sun Microsystems has announced plans to target additional platforms.
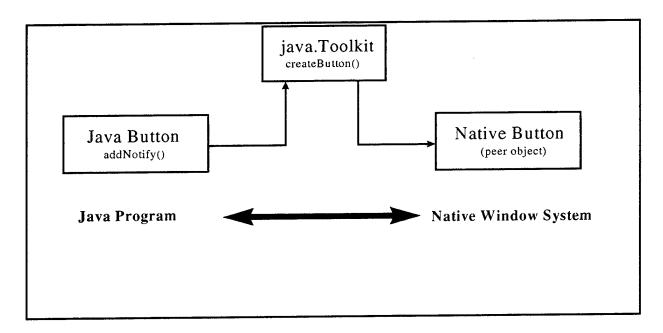
**Figure E.4   A java.awt.Toolkit creating a peer object for a specific platform**

When a component such as a Button, (Figure E.4), is first created and displayed to

the screen, the java.awt.Component class requests a java.awt.Toolkit class be created for

the corresponding peer object.  When you add a Button to a container, the container calls

the Button addNotify() method.  The addNotify() method then calls the java.awt.Toolkit

createbutton() method to make the Button's peer object in the native windowing system.

Therefore, the java.awt.Component class is able to reference the native windowing system

peer object and call the peer object directly [Exploring96].  Developers normally do not

directly work with the java.awt.Toolkit.  However, there are a few instances when

referencing a java.awt.Toolkit object is beneficial.   For instance, if a list of available fonts,

screen dimensions, or screen resolution, is needed then a reference to the java.awt.Toolkit

object is necessary.  This is achieved by calling the java.awt.Toolkit static method

getDefaultToolkit() and the desired methods in that object.  The use of the

java.awt.Toolkit and its implementation makes Java programs portable between all

platforms using  a Java developed java.awt.Toolkit interface.

## E. Summary

The Java AWT is a package of Java classes and interfaces allowing the creation of window-based, graphic user interface systems. The AWT provides mechanisms for graphics display, event management, text and graphics primitives, user interface components, and cross-platform layout. All of which make it impossible to predict how extensive, or for how long, Java will be used in WWW, multimedia, and information applications. Java is still in its infancy and although it provides a gentle introduction to standard software engineering issues. Java is currently not well-suited to perform or execute fast animation without native libraries, large scale development, text based programs, or standalone applications because of the cost associated with the overhead of the Java VM and the lack of a MVC model implementation. At the moment, commercial applications written in Java without native methods may not be able to compete with other applications on the market. However, in our opinion, given time, Java's solid design and engineering principles will prove to be valuable tools in creating GUI, Object-Oriented Framework applications. Although, the AWT is considered to be a GUI class hierarchy, Sun Microsystems plans to evolve Java into more of a framework that will better support rich applications.

Another issue of great importance is Java security. Although not discussed sufficiently in this chapter, the security issues surrounding Java applets/applications while running in a distributed environment need further attention. These security concerns center around the downloading of an applet via the web and the potential damage a malicious applet could perform on the client. In the current Java security model there are three fundamental issues: the design of the Java language, the standardized set of Java

libraries, and the Web browser that is used to view the Java applets. Until these issues receive further refinement and a formal verification of this model, there will be potential security risks.

# APPENDIX E-1  Panel and LayoutManager Example Source Code

```
import java.applet.Applet;  import java.awt.*;

class TestPanel extends Panel {
    Panel create(LayoutManager layout) {
        Panel SimplePanel = new Panel();
        SimplePanel.setLayout(layout);
        SimplePanel.add("North",  new Button("one"));
        SimplePanel.add("West",   new Button("two"));
        SimplePanel.add("South",  new Button("three"));
        SimplePanel.add("East",   new Button("four"));
        SimplePanel.add("Center", new Button("five"));
        return SimplePanel;
    }

    TestPanel() {
        setLayout(new CardLayout());
        add("one", create(new FlowLayout()));
        add("two", create(new BorderLayout()));
        add("three", create(new GridLayout(2, 2)));
        add("four", create(new BorderLayout(10, 10)));
        add("five", create(new FlowLayout(FlowLayout.LEFT, 10, 10)));
        add("six", create(new GridLayout(2, 2, 10, 10)));
    }

    public Dimension preferredSize(){
        return new Dimension(200, 100);
    }
}

public class PanelTest extends Applet {

    TestPanel cards;

    public PanelTest(){
        setLayout(new BorderLayout());
        add("Center", cards = new CardPanel());
        Panel SimplePanel = new Panel();
        SimplePanel.setLayout(new FlowLayout());
        add("South", SimplePanel);
        SimplePanel.add(new Button("first"));
        SimplePanel.add(new Button("next"));
        SimplePanel.add(new Button("previous"));
        SimplePanel.add(new Button("last"));
        Choice PickOne = new Choice();
        PickOne.addItem("one");
        PickOne.addItem("two");
        PickOne.addItem("three");
        PickOne.addItem("four");
        PickOne.addItem("five");
        SimpelPanel.add(PickOne);
    }
```

```java
public boolean action(Event evt, Object arg) {
        if (evt.target instanceof Choice) {
        ((CardLayout)cards.getLayout()).show(cards,(String)arg);
        } else {
        if ("first".equals(arg)) {
                ((CardLayout)cards.getLayout()).first(cards);
        } else if ("next".equals(arg)) {
                ((CardLayout)cards.getLayout()).next(cards);
        } else if ("previous".equals(arg)) {
                ((CardLayout)cards.getLayout()).previous(cards);
        } else if ("last".equals(arg)) {
                ((CardLayout)cards.getLayout()).last(cards);
        } else {
                ((CardLayout)cards.getLayout()).show(cards,(String)arg);
        }
        }
        return true;
    }

    public static void main(String args[]) {
        Frame a_frame = new Frame("PanelTest");
        PanelTest a_panel = new PanelTest();
        a_panel.init();
        a_panel.start();
        a_frame.add("Center", a_panel);
        a_frame.resize(280, 280);
            a_frame.show();
    }
}
```

# APPENDIX F - SOURCE CODE FOR VISIBILITY APPLET

The following is the source code for the Visibility example, as seen in chapter IV.

This applet demonstrates the scope and visibility of variables with in a Java applet.

Additionally, the sourcecode can be located at the web page address of http://vislab-

www.nps.navy.mil/~java/enhanced.

```
/*****************************************************
 * File: SamePackageSubclass.java
 * @author Dane Whitaker
 * @version 1.0
 * CS2973  Section 01
 * Access Modifiers Demonstration - Course Project
 * Operating Environment: Sun Sparc
 * Compiler: JDK
 * Description:   applet to illustrate access modifier effects for called
 *               instance variables in a subclass of a class in the same
 *               package
 *****************************************************/

import java.applet.*;
import java.awt.*;
import java.io.*;

public class SamePackageSubclass extends Applet (

Label  textLine1Label,
       textLine2Label,
       textLine3Label,
       textLine4Label,
       textLine5Label,
       textLine6Label,
       textLine7Label,
       textLine8Label,
       textLine9Label,
       textLine10Label;

TextField accessModifierField,
          canSeeField;

/*****************************************************
 * method: init
 * @return none
 * @param Parameter: none
 * Purpose: Init constructor for SamePackageSubclass Class
 *****************************************************/

public void init() (

   super.init();
   int bgcolor = 0xffffcc;    //lightyellow
   this.setFont(new Font("CourierNew", Font.PLAIN, 12));
   this.setBackground(new Color(bgcolor));

   // the layout for this applet is a gridbag
   GridBagLayout gridbag = new GridBagLayout();
   this.setLayout(gridbag);

   // create text text labels for to display text in body of applet
   textLine1Label = new Label("// A SOURCE FILE ");
   textLine2Label = new Label("package p;                    ");
   textLine3Label = new Label("class C (                     ");
   textLine4Label = new Label("          int i;              ");
   textLine5Label = new Label("                              ");
   textLine6Label = new Label("// IN ANOTHER SOURCE FILE ");
   textLine7Label = new Label("package p;                    ");
   textLine8Label = new Label("class D extends C (           ");
   textLine9Label = new Label("          int f( ) (return i;)");
   // can f see i ?  ");
   textLine10Label = new Label("                              ");

   // create access modifier and answer text fields
   accessModifierField = new TextField(15);
   accessModifierField.setBackground(new Color(bgcolor));
   accessModifierField.setFont(new Font("CourierNew", Font.BOLD, 12));
   accessModifierField.setEditable(true);

   canSeeField = new TextField(5);
   canSeeField.setBackground(new Color(bgcolor));
   canSeeField.setFont(new Font("CourierNew", Font.BOLD, 12));

   // associate a set of constraints with this applet and then add them
   constrain(this, textLine1Label, 0, 0, 3, 1, GridBagConstraints.WEST);
   constrain(this, textLine2Label, 0, 1, 3, 1, GridBagConstraints.WEST);
   constrain(this, accessModifierField, 0, 2, 1, 1, GridBagConstraints.WEST);
   constrain(this, textLine3Label, 1, 2, 3, 1, GridBagConstraints.WEST);
   constrain(this, textLine4Label, 0, 3, 3, 1, GridBagConstraints.WEST);
   constrain(this, textLine5Label, 0, 4, 3, 1, GridBagConstraints.WEST);
   constrain(this, textLine6Label, 0, 5, 3, 1, GridBagConstraints.WEST);
   constrain(this, textLine7Label, 0, 6, 6, 1, GridBagConstraints.WEST);
   constrain(this, textLine8Label, 0, 7, 6, 1, GridBagConstraints.WEST);
   constrain(this, textLine9Label, 0, 8, 6, 1, GridBagConstraints.WEST);
   constrain(this, canSeeField, 7, 8, 1, 1, GridBagConstraints.WEST);
   constrain(this, textLine10Label, 0, 9, 6, 1, GridBagConstraints.WEST);

)

/*****************************************************
 * method: constrain
 * @return none
 * @param Parameter:   Container container, Component comp, int grid_x,
 *                     int grid_y, int grid_width, int grid_height,
 *                     int anchor
 * Purpose:  Calls another constrain method above parameters and adds
 *           parameters for the GridBagContraints object weight attributes
 *****************************************************/

private void constrain(Container container, Component comp,
                       int grid_x, int grid_y,
                       int grid_width, int grid_height,
                       int anchor)(

   constrain(container, comp, grid_x, grid_y,
             grid_width, grid_height,
             anchor, 0.0, 0.0);

)

/*****************************************************
 * method: constrain
 * @return: none
 * @param Parameter:   Container container, Component comp, int grid_x,
```

```
 *             int grid_y, int grid_width, int grid_height,
 *             int anchor, double weight_x, double weight_y
 * Purpose:   Intantiates a GridBagConstraints object and assigns parameters to
 *            GridBagConstraints object attributes
 *************************************************************************/

private void constrain(Container container, Component comp,
                       int grid_x, int grid_y,
                       int grid_width, int grid_height,
                       int anchor, double weight_x, double weight_y) {

    GridBagConstraints c = new GridBagConstraints();
    c.gridx = grid_x;  c.gridy = grid_y;
    c.gridwidth = grid_width;  c.gridheight = grid_height;
    c.fill = GridBagConstraints.NONE;
    c.anchor = anchor;
    c.weightx = weight_x;
    c.weighty = weight_y;
    c.insets = new Insets(0, 0, 5, 0);
    ((GridBagLayout) container.getLayout()).setConstraints(comp, c);
    container.add(comp);
}


/*************************************************************************
 * method:  action
 * @return: boolean
 * @param Parameter:  Event event, Object arg
 * Purpose:  event handler applet actions
 *************************************************************************/

public boolean action(Event event, Object obj) {

    if(event.target == accessModifierField) {
        lookupVisibility();
        return true;
    }

    return false;
}


/*************************************************************************
 * method: lookupVisibility
 * @return: none
 * @param Parameter:  none
 * Purpose:  determines visibility based on access modifier string in the
 *           accessModifierField and places Yes, No, or Error in the
 *           canSeeField
 *************************************************************************/

private void lookupVisibility() {

    if (accessModifierField.getText().equals("") ||
        accessModifierField.getText().equals("public") ||
        accessModifierField.getText().equals("protected")) {

        canSeeField.setText("Yes");

    }
    else if (accessModifierField.getText().equals("private protected") ||
             accessModifierField.getText().equals("private")) {
```

```
            canSeeField.setText("No");
        }
        else {

            canSeeField.setText("error");
        }
    }
}
```

# APPENDIX G - SOURCE CODE FOR COURSE SERVER

The Course Server is a multithreaded application running on a computer that implements a reliable stream network connection between the Course Server and the QuestionApplet Client (Appendix H).

The following QuestionApplet Client and Course Server process diagram is provided for further clarification of the source code in Appendix G & H. The QuestionApplet Client process and the Course Server process are separated by a dotted line.
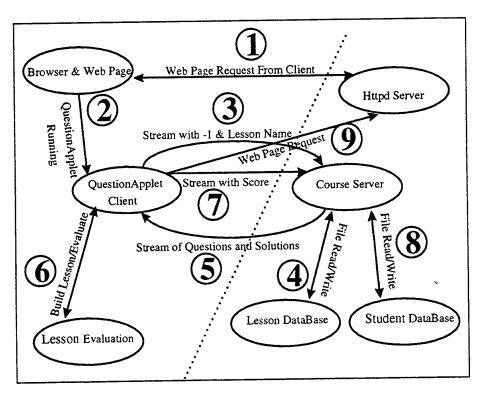


**Figure G.1. QuestionApplet Client and Course Server Process Diagram**

1. The Web browser makes a request to the Httpd Server which sends the Web page, with the QuestionApplet bytecodes in it, back to the Web browser. The

Web browser then translates the HTML and Java bytecodes to create the QuestionApplet
Client.

2.    The QuestionApplet Client is executed and builds an "Evaluation
Button" at the bottom of the Web Page.  This button, when clicked on, will create the Self
Evaluation form for the lesson.

3.    The Course Server uses the ServerSocket class to accept
connections from the QuestionApplet Client.  When a QuestionApplet Client connects to
the specified port, that the Course Server is listening on, the ServerSocket allocates a new
Socket object, and a separate thread is created for the QuestionApplet Client to
communicate with.  When the connection between the QuestionApplet Client and the
Course Server is first established the QuestionApplet Client sends a stream to the Course
Server with a "-1" flag, signifying that this is a new lesson, and the name of the lesson file
to load from the lesson database.

4.    The Course Server then parses the input stream that is sent from
the QuestionApplet Client and opens the requested lesson file from the Lesson Database.
The lesson file is then read into the Course Server and then output to the QuestionApplet
Client via the java.io.DataInputStream and java.io.DataOuput classes.  The
java.io.DataInputStream and the java.io.DataOuput classes are used because they
allow the reading and writing of lines of text and Java primitive data types in a machine-
independent format.

5.    The Course Server then sends the lesson data to the
QuestionApplet Client via the java.io.DataOutput class and returns to listening for
other QuestionApplet Clients trying to connect.

6. The QuestionApplet Client receives and parses the input stream from the Course Server. The QuestionApplet Client then builds the evaluation questions for the desired lesson.

7. Once the user has completed the self evaluation, the QuestionApplet Client sends a stream consisting of the users score to the Course Server, however without the "-1" flag.

8. If the "-1" flag is not received by the Course Server then the Course Server understands that the data in the stream from the QuestionApplet Client is the students score from the QuestionApplet Client self evaluation. The Course Server then stores the score to the Student Database and returns to listening for other QuestionApplet Clients attempting to connect.

9. Once the score has been sent to the Course Server the QuestionApplet uses the "Next.html" field from the initially parsed input stream to request the next HTML file lesson from the Httpd Server. Note that the Course Server is multithreaded and the Server object itself is a thread. This allows the Course Server to loop forever, processing and listening for QuestionApplet Client requests and sending HTML requests to the Httpd Server. Additionally, the Course Server must be running with a Httpd Server in parallel else the QuestionApplet Client will not be able to establish communications with the Httpd Server nor the Course Server.

```
/***********************************************************
 * FS3.java
 *
 * Remarks:
 *
 * modified by Rich Moormann
 *
 * This example is from the book _Java in a Nutshell_ by David Flanagan.
 * Written by David Flanagan. Copyright (c) 1996 O'Reilly & Associates.
 * You may study, use, modify, and distribute this example for any purpose.
 * This example is provided WITHOUT WARRANTY either expressed or implied.
 ***********************************************************/

//** imported packages
import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

/***********************************************************
 * FS3 method
 *
 * Remarks: FS3 is a java application that binds to a port then
 * waits for clients to connect. A new thread is created to handle
 * each connection.
 ***********************************************************/
public class FS3 extends Thread {
    public final static int DEFAULT_PORT = 6777;
    protected int port;
    protected ServerSocket listen_socket;

    // Exit with an error message, when an exception occurs.
    public static void fail(Exception e, String msg) {
        System.err.println(msg + ": " + e);
        System.exit(1);
    }

    // Create a ServerSocket to listen for connections on;  start the thread.
    public FS3(int port) {
        if (port == 0) port = DEFAULT_PORT;
        this.port = port;
        try { listen_socket = new ServerSocket(port); }
        catch (IOException e) { fail(e, "Exception creating server socket"); }
        System.out.println("FS3: listening on port " + port);
        this.start();
    }

    // The body of the server thread. Loop forever, listening for and
    // accepting connections from clients.  For each connection,
    // create a Connection object to handle communication through the
    // new Socket.
    public void run() {
        try {
            while(true) {
                Socket client_socket = listen_socket.accept();
                Connection c = new Connection(client_socket);
            }
        }
        catch (IOException e) {
            fail(e, "Exception while listening for connections");
        }
    }

    // Start the server up, listening on an optionally specified port
    public static void main(String[] args) {
        int port = 0;
        if (args.length == 1) {
            try { port = Integer.parseInt(args[0]); }
            catch (NumberFormatException e) { port = 0; }
        }

        // System.out.println("Server Running on Port# "+ port);
        new FS3(port);
    }
}

/***********************************************************
 * Connection class
 *
 * Remarks: This class is the thread that handles all communication with a
 * client
 ***********************************************************/
class Connection extends Thread {
    protected Socket client;
    protected DataInputStream DISin;   // THis will be the stream from the client
    protected PrintStream out;

    // Initialize the streams and start the thread
    public Connection(Socket client_socket) {
        client = client_socket;
        System.out.println("in Connection, after client= " +client);

        try {
            DISin = new DataInputStream(client.getInputStream());
            out = new PrintStream(client.getOutputStream());
        }
        catch (IOException e) {
            try { client.close(); } catch (IOException e2) { ; }
            System.err.println("Exception while getting socket streams: " + e);
            return;
        }
        this.start();
    }

    // Provide the service.
    // Read a string from the applet, read a text file into line, send it to applet
    public void run() {
        String line, FromAppletLine=null;
        String next_file;       // will be defined as an initial file
        int score_value;

        System.out.println("in Connection run()");

        // ******* get data from the applet
        // ** applet must tickle the server to get the first file across
        try {
            for(;;) {
                // read in a line
                FromAppletLine = DISin.readLine();
                if (FromAppletLine != null) break;
                System.out.println("in Connection FromAppletLine = "+FromAppletLine);
            }//for
        }//try
        catch (IOException e) {
            next_file="data.txt";
        }
```

```
      // Line has been read from applet, now process it.

          System.out.println("in Connection from  applet string = "+FromAppletLine );

      // PUT PARSING CODE HERE
          System.out.println("In FS3 parse_incoming");

          StringTokenizer t = new StringTokenizer(FromAppletLine, "|");
          score_value = Integer.valueOf(t.nextToken()).intValue();
          next_file = t.nextToken();

          System.out.println(" next_file="+next_file);
          System.out.println(" score_value= "+score_value);
          if (score_value < 0) {
               System.out.println("  no  score_value reported = "+score_value);
          }

      // END PARSING CODE


      // ***** added from Editor
      // ***** This all reads the local file and sends it to the client

          FileInputStream FISin = null;          //This is the stream from the local files
          DataInputStream dataIn = null;
          BufferedInputStream bis = null;

      // ***** we have the filename now get and push the data

          StringBuffer outgoingBuffer = new StringBuffer();
          try {
              FISin = new FileInputStream(next_file);
              bis = new BufferedInputStream(FISin);
              dataIn = new DataInputStream(bis);
          }catch(Throwable e) {
              System.out.println("Can't open \""+next_file+"\"");
          }

          try {
              while ((line = dataIn.readLine()) != null) {
                  outgoingBuffer.append(line);
              }
              FISin.close();
              out.println(outgoingBuffer); //send it!
              System.out.println(outgoingBuffer); // what was sent

          } catch(IOException e) {
              System.out.println("Can't read \""+next_file+"\"");
          }


      }//run
}//connection
```

211

# APPENDIX H - SOURCE CODE FOR QUESTIONAPPLET CLIENT

The QuestionApplet Client provides the GUI for the enhanced evaluation application as referenced in Chapter IV. Additionally, the QuestionApplet Client sends and receives data from the Course Server creating an interactive evaluation form. The QuestionApplet Client and Course Server Process diagram in Appendix G is provided for further clarification of the source code in Appendix H. The following source code was implemented to support the QuestionApplet Client portion of the enhanced evaluation application.

```
e;re;re;re;r; re;er;r;rty rSky?|3|Red|Blue|Black|2"+
                  "|What color is the Sky?|3|Red|Blue|Black|2"+
                  "|What color is the Sky?|3|Red|Blue|Black|2"+
                  "|What color is the Sky?|3|Red|Blue|Black|2";
";

/**********************************************
* init method
*
* Remarks:   Start applet and build questions
*
* Incoming Format:
*
* # of questions|background_color|foreground_color|next_file|rem_file|
*   |question|#choices|choice array|solution#
**********************************************/
public void init() {

    super.init();                                      //** call super init method
    lesson_file = getParameter("lesson_file");  //** get lesson_file name
    //** send -1 to server 1st time to init as 1st time
    score = -1;
    send_stream(lesson_file,score);  //** get the lesson file name
                                     //** passed from HTML page

    score = 0;
    build_button();                 //** build applet button
    build_questions();              //** build question frames
    setLayout(null);                //** set layout = null
    addNotify();
    resize(160,34);                 //** button resizing

} //** end of init

/*******************************************************************
* init method
*
* Remarks: Overload with the new lesson file name to load
*******************************************************************/
public void init(String lesson_file, int score) {
    send_stream(lesson_file,score);  //** get the lesson file name passed
                                     //** from HTML page
    build_questions();               //** build question frames
} //** end of init

/*******************************************************************
* send_stream method
*
* Remarks: sends lesson file name stream to the java server
*******************************************************************/
public void send_stream(String lesson_file, int score) {

    this.lesson_file = lesson_file;
    this.score       = score;

    try {
        s = new Socket(this.getCodeBase().getHost(), PORT);
        out = new PrintStream(s.getOutputStream());
        out.println("score"+"|"+lesson_file);
        System.out.println("Score|lesson_file-->"+score+"|"+lesson_file);
    }
    catch (IOException e) {
        System.out.println("No Socket->"+e.toString());
        error("Socket Could Not Be Opened!");
```

```
/**********************************************
* QuestionApplet class
*
* @version 1.0, QuestionApplet
* @author Wes Hester & Rich Moormann, Naval Postgraduate School, Monterey, CA
*
*
* Remarks:  This applet connects to the Java server application, the
* feeds the QuestionApplet a stream of lesson questions. The QuestionApplet
* Applet then dynamically generates questions and possible solutions to the
* QuestionApplet. The QuestionApplet grades the questions. If the grade is
* pass then the applet calls the next HTML page, else a remidial HTML page is
* called.
*
* Last Update: 15 Feb 97
**********************************************/

/*********************************
* imported packages
*********************************/
import java.awt.*;
import java.applet.*;
import java.io.*;
import java.net.*;
import java.util.*;

/*********************************************
* QuestionApplet class
*
* Remarks: main class
*********************************************/
public class QuestionApplet extends Applet {

    //** declarations
    Button button;                      //** self evaluation button
    create_layout layout;               //** array of frames
    String    rem_file;                 //** remedial html file name
    String    next_file;                //** next file name to grab
    String    next_html;                //** next html file to goto
    public int score   = 0;             //** keeps score of lessons
    public static final int PORT = 6777; //** physical port on server
    Socket s;                           //** socket to open/close
    DataInputStream in;                 //** input stream to read
    PrintStream out;                    //** output stream to send
    String line;                        //** input stream handler string
    String lesson_file;                 //** passed from HTML file
    summary summary_object;

/**********************************************
* Sample Color codes:
*
* Magenta 16711935, Red 16711680, Yellow 16776960
* Green 65280, Blue 255, White 1677215, Black 0
**********************************************/

/* used for testing
String line = "4|16711680|65289|next.txt|remidial.txt|next.html"+
              "|What color is an apple?|5|Orangish erwe we e ee colored or this woudl d
tsdfdsfsd dsadfsfsfsfssfsd|Blue|Black|Green|Green|4"+
              "|What color is a Car?|4|Pink|Blue|Black|Green|4"+
              "|What color is the Night Sky wer wsdfas sdfsdsdfsdfsd fsdf sdfsd sdsdf sd
fsadfsdfsdt sdt s?|3|Red|Blue|Black|2"+
              "|What color is the Day rty rty rty yyrteeyeyery rty ryrty yr;rey rey reyr
```

```java
        }
) //** end of send_stream

/*********************************************
* build_button method
*
* Remarks: put the Self Eval button in the web page
**********************************************/
public void build_button() {
    if (button == null) {
        button = new java.awt.Button("Self Evaluation");
        button.reshape(0,0,160,34);
        button.setFont(new Font("TimesRoman", Font.BOLD, 18));
        add(button);
    }
} //** end of build_button

/*********************************************
* build_questions class
*
* Remarks: build the questions frames
**********************************************/
public void build_questions() {

    //** get the input stream from the server
    line = get_stream();

    //** dynamically create question frames and load in array
    layout = new create_layout(line,
                    next_file,
                    next_html,
                    score,
                    layout,
                    s,
                    out,
                    PORT,
                    this,
                    lesson_file,
                    summary_object);

    layout.setResizable(false);

} //** end of build questions

/*********************************************
* get_stream method
*
* Remarks: Create a socket to communicate with a server on port 6789
* of the host that the applet's code is on. Then create streams to use with
* the socket. Then create a TextField for user input and a TextArea
* for server output. Finally, create a thread to wait for and
* display server output
**********************************************/
public String get_stream() {
    System.out.println("Before try in get_stream");

    try {
        in = new DataInputStream(s.getInputStream());
        System.out.println("In get_stream");

        line = get();
        System.out.println("Connected to "
                + s.getInetAddress().getHostName()
                + ":" + s.getPort());
    }

    catch (IOException e) {
        error("Error in Reading Lesson From Server!");
        System.out.println(e.toString());
    }

    return (line);

} //** end of get_stream

/*********************************************
* get method
*
* Remarks: reads the stream from the server
**********************************************/
public String get() {
    int num_trys = 0;

    try {

        //** wait until it gets a stream from server
        for(;;) {
            line = in.readLine(); //** read in string line from server
            System.out.println("for loop line ="+line);
            if (line != null) break;
            num_trys = (num_trys +1);

            //** build a dialog to ask whether to abort or continue...
            if (num_trys > 80) {
                System.out.println("Error..Abort or Wait");
                error("Nothing coming in from server");
            }
        } //** for

    } //** try
    catch (IOException e) {
        System.out.println("Error");
        error("Error Reading Lesson From File!");
    } //** if
    finally {
        try {
            if (s != null) {
                s.close();
                System.out.println("Connection closed by Applet");
            }
        }
        catch(IOException e2) {
            System.out.println("Error");
            error("Socket Could Not Be Closed!");
        }
    } //** finally

    return (line);

} //** get method
/*********************************************
```

# QuestionApplet.java

```java
* error method
*
* Remarks: display error dialog
*********************************************/
private void error (String message) {
   popup popupBox;
   popupBox = new popup("Error Dialog",message);
   popupBox.show();
} //** end of error method

/*********************************************
* handleEvent method
*********************************************/
public boolean handleEvent(Event evt) {

   System.out.println("in handle event of Applet");
   switch(evt.id) {
      case Event.ACTION_EVENT: {
         if (evt.target == button) {
            //** show question frames
            layout.show();
         }

      } //** end of case

   } //** end of switch

   return(true);

} //** end of handleEvent

} //** end of QuestionApplet class


/*********************************************************************
* create_layout class
*
* Remarks: Create the array of Frames
*********************************************************************/
class create_layout extends Frame (

KeyPressManagerPanel keyPressManagerPanel;
ScrollingPanel scrollingPanel1;
String next_file;
java.awt.Choice    choice[];
java.awt.Label     label[];
java.awt.Button    button1,button2;
java.awt.TextArea  questionarea[];
Label              copyright;
int score;
int num_questions;
create_layout layout;
PrintStream out;
Socket s;
int PORT;
QuestionApplet app;
String next_html;
int token_position = 0;
String line;
int num_choices      []; //** array of choices
int solution_num     []; //** array of solutions
```

```java
String question       []; //** questions
String choice_array   [] []; //** puts spaces between each question
int buf = 190;
String lesson_file;
int background_color;
int foreground_color;
String rem_file;
summary summary_object;
int choice_len = 150;
int counter;
int magic_width;
StringBuffer buffer;

/*********************************************************************
* create_layout method
*
* Remarks:  constructor to create layout frames for each question
*********************************************************************/
create_layout(String line,
              String next_file,
              String next_html,
              int score,
              create_layout layout,
              Socket s,
              PrintStream out,
              int PORT,
              QuestionApplet app,
              String lesson_file,
              summary summary_object) {

   this.lesson_file = lesson_file;
   setTitle("Question Frame For Lesson "+lesson_file);
   setLayout(null);

   addNotify();
   resize(624,524);           //** frame size
   setResizable(false);

   //** pass object pointer
   this.layout      = layout;
   this.next_file   = next_file;
   this.out         = out;
   this.s           = s;
   this.PORT        = PORT;
   this.app         = app;
   this.next_html   = next_html;
   this.score       = score;
   this.line        = line;
   this.summary_object = summary_object;

   //** get init values from stream (#questions, colors...
   line = get_init_vars(line);
   //setCursor(Frame.DEFALUT_CURSOR);    //** make cursor a clock

   //** allocate componet arrays
   allocate_arrays(num_questions);

   //** create scrollable panel
   scrollingPanel1 = new ScrollingPanel();
   scrollingPanel1.setLayout(null);
   scrollingPanel1.reshape(20,40,600,482);
   scrollingPanel1.setBackground(new Color(background_color));
```

```java
add(scrollingPanel1);

//** create keypressmanager panel
keyPressManagerPanel1 = new KeyPressManagerPanel1();
keyPressManagerPanel1.setLayout(null);
keyPressManagerPanel1.reshape(0,0,590,(290+(buf*num_questions)));
keyPressManagerPanel1.setBackground(new Color(background_color));
scrollingPanel1.add(keyPressManagerPanel1);

//** calculate magic_width dependent on which os.
/* Font f = new Font("Dialog", Font.BOLD, 14);

FontMetrics fm = getFontMetrics(f);
magic_width   = fm.getMaxAdvance();
System.out.println("Font Width="+magic_width);
*/

//** loop until no more questions, adding to panel
load_panel_components(line);

//** button for Canceling lesson grade
button1 = new java.awt.Button(" Cancel ");
button1.reshape(135,(70+(buf*num_questions)),94,40);
button1.setFont(new Font("Dialog", Font.BOLD, 18));
button1.setForeground(new Color(foreground_color));
keyPressManagerPanel1.add(button1);

//** button for submitting grade
button2 = new java.awt.Button(" Grade ");
button2.reshape(305,(70+(buf*num_questions)),94,40);
button2.setFont(new Font("Dialog", Font.BOLD, 18));
button2.setForeground(new Color(foreground_color));
keyPressManagerPanel1.add(button2);

//** footer message for copyright
copyright = new java.awt.Label(" Copyright 1997, Naval "+
                "Postgraduate School, CS Dept.");
copyright.reshape(80,(155+(buf*num_questions)),360,40);
copyright.setFont(new Font("Dialog", Font.BOLD, 12));
copyright.setForeground(new Color(foreground_color));
keyPressManagerPanel1.add(copyright);

} //** end of create_layout method

/*****************************************************************
* load_panel_componets method
*
* Remarks: load the Question Label, TextArea with question, and choice for
*          each question
*****************************************************************/
void load_panel_components(String line) {

    int token_position = 0;
    int num = 0;

    for (int i = 0; i < num_questions; i++) {

        //** question number label
        num = (i+1);
        label[i] = new java.awt.Label("Question #"+num);
        label[i].reshape(15,(22+(buf*i)),110,30);
```

```java
label[i].setFont(new Font("Dialog", Font.BOLD, 14));
label[i].setForeground(new Color(foreground_color));
keyPressManagerPanel1.add(label[i]);

//** get the questions
StringBuffer buffer = new StringBuffer();

token_position = line.indexOf((int)'|');
question[i] = line.substring(0,token_position);
line = line.substring(token_position+1,line.length()); // adjust

//** build questions into textareas
questionarea[i] = new java.awt.TextArea();

//** put the question into a stringbuffer for \n inserts
buffer.append(' '+question[i]);

magic_width = 62;  //** number of characters per line

counter = magic_width;

//** calculate line <cr> into buffer
while (counter < buffer.length()) {

    if (buffer.charAt(counter) != ' ') {
        //** then backup until we find a blank space

        while (buffer.charAt(counter) != ' ') {
            counter = (counter - 1); //** back up
        }
    }
    //** insert the <cr>
    buffer.insert(counter,"\n");
    counter = counter + magic_width;
}

//** layout questions in questionarea
questionarea[i].setText(buffer.toString());
questionarea[i].setEditable(false);
questionarea[i].reshape(22,(67+(buf*i)),535,74);
questionarea[i].setFont(new Font("Dialog", Font.BOLD, 14));
questionarea[i].setForeground(new Color(foreground_color));
questionarea[i].setBackground(new Color(16777215)); //** white
keyPressManagerPanel1.add("Center",questionarea[i]);

//** get # choices
token_position = line.indexOf((int)'|');
if (token_position == 0)
    token_position = line.length()+1;
num_choices[i] = Integer.valueOf(
        line.substring(1,token_position-1)).intValue();
if (token_position != line.length())
    line = line.substring(token_position+1,line.length()); // adjust

//** load choice array
for (int j = 0; j < num_choices[i]; j++) {
    //** get the choices into array
    token_position = line.indexOf((int)'|');
    choice_array[i][j] = line.substring(0,token_position);
    //** find the longest possible choice
    if ((choice_len < choice_array[i][j].length()) &&
```

```java
**********************************************************;
String get_init_vars(String line) {

    //** get the # questions
    token_position = line.indexOf((int)'|');
    num_questions = Integer.valueOf(
                line.substring(0,token_position).intValue();
    line = line.substring(token_position+1,line.length());

    //** get background_color
    token_position = line.indexOf((int)'|');
    background_color = Integer.valueOf(
                line.substring(0,(token_position)).intValue();
    line = line.substring(token_position+1,line.length()); // adjust line
    setBackground(new Color(background_color));

    //** get foreground_color
    token_position = line.indexOf((int)'|');
    foreground_color = Integer.valueOf(
                line.substring(0,(token_position)).intValue();
    line = line.substring(token_position+1,line.length()); // adjust line
    setForeground(new Color(foreground_color));

    //** get the next file name
    token_position = line.indexOf((int)'|');
    next_file = line.substring(0,token_position);
    line = line.substring(token_position+1,line.length());

    //** get the remidial file name
    token_position = line.indexOf((int)'|');
    rem_file = line.substring(0,token_position);
    line = line.substring(token_position+1,line.length());

    //** get the next.html name
    token_position = line.indexOf((int)'|');
    next_html = line.substring(0,token_position);
    line = line.substring(token_position+1,line.length());

    return(line);

} //** end of get_init_vars

/********************************************************
* handleEvent method
********************************************************/
public boolean handleEvent (Event evt) {

    System.out.println("in handle event of createlayout");
    switch(evt.id) {
        case Event.ACTION_EVENT: {

            if (evt.target == button1) {
                hide();
                dispose();
                System.gc();
            }

            //** if button2 pressed and if not alreay pushed
            else if ((evt.target == button2) && (summary_object == null)) {
                hide();
                dispose();
                System.gc();
                //** calculate score, if they selected then add 10
```

```java
            (choice_array[i][j].length() < 300))
        choice_len = (choice_array[i][j].length());
    line = line.substring(token_position+1,line.length()); // adjust
    } //** end of for num_choices

    choice[i] = new java.awt.Choice();
    keyPressManagerPanel1.add(choice[i]);
    choice[i].reshape(200,(148+(buf*i)),choice_len,30);
    choice[i].setFont(new Font("Dialog", Font.BOLD, 14));
    choice[i].setForeground(new Color(foreground_color));
    choice[i].setBackground(new Color(16777215)); //** white
    for (int v = 0; v < num_choices[i]; v++) {
        choice[i].addItem(choice_array[i][v]);
    }

    //** get solution num
    token_position = line.indexOf((int)'|');
    if (line.length() == 1) {
        solution_num[i] = Integer.valueOf(line).intValue();
    }
    else {
        solution_num[i] = Integer.valueOf(line.substring(1,token_position-1)).intValue()
    }

    if (token_position != line.length())
        line = line.substring(token_position+1,line.length()); // adjust line
    }

} //** end of for loop

} //** end of load_panel_components

/***********************************************
* clickedhelp method
***********************************************/
public void clickedhelp () {
    help HelpObject;
    HelpObject = new help();
    HelpObject.show();
} //** end of clickedHelp

/***********************************************
* allocate_arrays method
*
* Remarks: create arrays for screen componets
***********************************************/
void allocate_arrays(int num_questions) {
    //** allocate arrays
    num_choices    = new int       [num_questions];
    solution_num   = new int       [num_questions];
    question       = new String    [num_questions];
    label          = new Label     [num_questions];
    questionarea   = new TextArea  [num_questions];
    choice         = new Choice    [num_questions];
    choice_array   = new String    [num_questions] [5];
} //** end of allocate_arrays method

/***********************************************
* get_init_vars method
*
* Remarks: get the initial values from the input stream
```

```
        for (int j = 0; j < num_questions; j++) {
            if (solution_num[j] == choice[j].getSelectedIndex()+1)
                score = score + 10;
        }

        //** build summary object if not already built
        summary_object = new summary(next_file,
                                     next_html,
                                     s,
                                     out,
                                     PORT,
                                     app,
                                     score);

        summary_object.show(); //** display object

    ) //** end of button2 if

    ) //** end of case
    //** help right mouse clicked
    case Event.MOUSE_DOWN: {
        if (evt.modifiers == Event.META_MASK) {
            clickedhelp();
        }
        break;
    ) //** end of case

    case Event.WINDOW_DESTROY: {
        hide();
        dispose();
        break;
    }

    ) //** end of switch

    return(true);

) //** end of handleEvent

) //** end of create_layout class

/*****************************************************************
* summary class
*
* Remarks: build the summary frame to display results of evaluation
*****************************************************************/
class summary extends Frame {

KeyPressManagerPanel keyPressManagerPanel;
java.awt.Choice   choice;
java.awt.Label    label;
java.awt.Label    label2;
java.awt.Button   button;
java.awt.TextArea questionarea;
int score;
int solution_num;
int i;
int not_scored_yet = 0;
String next_file;
String next_html;
PrintStream out;
Socket s;
```

```
int PORT;
QuestionApplet app;

/*****************************************************************
* summary method
*
* Remarks: construct for summary frame
*****************************************************************/
summary(String next_file, String next_html, Socket s, PrintStream out,
        int PORT, QuestionApplet app, int score) {

    setTitle("Evaluation Summary");

    setLayout(null);
    addNotify();
    resize(426,266);

    //** pass pointers to ref obj
    this.score        = score;
    this.solution_num = solution_num;
    this.i            = i;
    this.next_file    = next_file;
    this.out          = out;
    this.s            = s;
    this.PORT         = PORT;
    this.app          = app;
    this.next_html    = next_html;

    //** build the componets
    keyPressManagerPanel1 = new KeyPressManagerPanel();
    keyPressManagerPanel1.setLayout(null);
    keyPressManagerPanel1.reshape(0,0,427,270);
    keyPressManagerPanel1.setBackground(new Color(16776960));
    add(keyPressManagerPanel1);

    //** label for Evaluation Grade
    label = new java.awt.Label("Evaluation Grade Summary");
    label.reshape(100,60,230,30);
    label.setFont(new Font("Dialog", Font.BOLD, 14));
    keyPressManagerPanel1.add(label);

    //** your score is label
    if (score < 0) score =0;
    label2 = new java.awt.Label("Your current score is \""+score+"\"");
    label2.reshape(100,100,240,30);
    label2.setFont(new Font("Dialog", Font.BOLD, 14));
    keyPressManagerPanel1.add(label2);

    //** add next lesson button
    button = new java.awt.Button(" Next Lesson ");
    button.reshape(150,195,130,40);
    button.setFont(new Font("Dialog", Font.BOLD, 18));
    button.setForeground(new Color(0));
    keyPressManagerPanel1.add(button);

) //** end of create_layout method

/*****************************************************************
* clickedhelp method
*****************************************************************/
public void clickedhelp () {
    help HelpObject;
    HelpObject = new help();
```

```java
    HelpObject.show();
}

/*************************************************
* send_stream method
*
* Remarks:  sends lesson file name stream to the java server
*************************************************/
public void send_stream(String next_file, String next_html,
                QuestionApplet app,
                int PORT, PrintStream  out,int score)  {

try {  //** call next html page
    AppletContext context = app.getAppletContext();
    //URL u = new URL(app.getCodeBase()+next_html);
    URL u = new URL(next_html);

    context.showDocument(u, "upper"); //** show browser

    //** call mr, garbage collector
    System.gc();

    //** call next lesson file to load
    app.init(next_file,score);

}

catch (Exception e) {
    System.out.println("No Socket->"+e.toString());
    popup popupBox;
    popupBox = new popup("Error Dialog","Error Referencing "+
                    "HTML Page");

    popupBox.show();
}

} //** end of send_stream


/*************************************************
* handleEvent method
*************************************************/
public boolean handleEvent(Event evt) {

System.out.println("in handle event of createlayout");
switch(evt.id) {
    case Event.ACTION_EVENT: {
        if (evt.target == button) {
            hide();       //** hide the Frame
            dispose();       //** free the system resources
            send_stream(next_file,next_html,app,PORT,out,score);
        }
    } //** end of case
    //** help right mouse clicked
    case Event.MOUSE_DOWN: {
        if (evt.modifiers == Event.META_MASK) {
            clickedhelp();
        }

        break;
    } //** end of case
    case Event.WINDOW_DESTROY: {
        hide();
        dispose();
        break;
    }
    } //** end of switch

    return(true);

} //** end of handleEvent

} //** end of summary class


/*************************************************
* popup class
*
* Remarks: This class pops up online of text and provides a close
*          button.
*************************************************/
class popup extends Frame  {

Button button1;
Label Main_Label;
Color Keywordcolordefault = Color.red;

/*************************************************
* Remarks: Constructor for class
*************************************************/
public popup(String title, String message)  {

super(title);
setLayout(null);
addNotify();
resize(insets().left + insets().right + 406, insets().top +
            insets().bottom + 120);

setCursor(Frame.HAND_CURSOR); //** make cursor a hand

this.setForeground(Color.red);
this.setBackground(Color.yellow);
Main_Label=new Label(message, Label.CENTER);
Main_Label.setFont(new Font("Dialog",Font.BOLD,14));
add(Main_Label);
Main_Label.reshape(5,45,400,32);

button1=new Button("Close");
add(button1);
button1.reshape(insets().left + 150,insets().top + 75,100,23);

show();

} //** end of popup

/*************************************************
* clickedhelp method
*************************************************/
public void clickedhelp () {
    help HelpObject;
    HelpObject = new help();
    HelpObject.show();
}
```

```
/*******************************************************
* Message()
*
* Remarks:  This method outputs Message and yes or no.
*
*******************************************************/
public void Message(int txcolor,String title, String message) {
    MessageBox theMessageBox;
    theMessageBox = new MessageBox(txcolor,this,title,message,
                            Color.red);

    theMessageBox.show();
} //end of Message

public synchronized void show() {
    move(50, 50);
    super.show();
}

public boolean handleEvent(Event event)  {

    if (event.id == Event.ACTION_EVENT && event.target == button1)  {
        clickedButton1();
        return true;
    }
    else

    if (event.id == Event.WINDOW_DESTROY)  {
        hide();          // hide the Frame
        dispose();       // tell windowing system to free resources
        return true;
    }

    //Right mouse clicked
    else if (event.id == Event.MOUSE_DOWN) {

        if (event.modifiers == Event.META_MASK)  {
            clickedhelp();
        }
    }

    return super.handleEvent(event);
}

public void clickedButton1()  {
    handleEvent(new Event(this, Event.WINDOW_DESTROY, null));
}

} //** end of popup class


/*******************************************************
* MessageBox class
*
* This class is a basic extension of the Dialog class.  It can be used
* by subclasses of Frame.  To use it, create a reference to the class,
* then instantiate an object of the class (pass 'this' in the constructor),
* and call the show() method.
*
* example:
*
* MessageBox theMessageBox;
*         theMessageBox = new MessageBox(this);
*         theMessageBox.show();
*
*******************************************************/
class MessageBox extends Dialog  {

    protected Button button;
    protected Label label;
    int txcolor;

    public MessageBox(int txcolor,
                    popup parent,
                    String title,
                    String message,
                    Color Keywordcolordefault)  {

        //** Create a dialog with the specified title
        super(parent, title, false);
        setResizable(false);

        //** Create and use a BorderLayout manager with specified margins
        this.setLayout(new BorderLayout(15, 15));
        resize(insets().left + insets().right + 261, insets().top + insets().bottom

        //** standard prefs for dialogs
        setFont(new Font("TimesRoman", Font.PLAIN, 12));
        setForeground(Color.black);
        setBackground(Keywordcolordefault);

        //** change text color to red
        this.txcolor = txcolor;
        if (txcolor == 2) setForeground(Color.red);
        else if (txcolor == 3) setForeground(Color.yellow);

        //** Create the message component and add it to the window
        label = new Label(message);
        this.add("Center", label);

        //** Create an Okay button in a Panel; add the Panel to the window
        //** Use a FlowLayout to center the button and give it margins.
        button = new Button("Okay");
        Panel p = new Panel();
        p.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 15));
        p.add(button);
        add("South", p);

        //** Resize the window to the preferred size of its components
        this.pack();
    }

    public MessageBox(int txcolor,
                    create_layout parent,
                    String title,
                    String message,
                    Color Keywordcolordefault)  {

        //** Create a dialog with the specified title
        super(parent, title, false);
        setResizable(false);

        //** Create and use a BorderLayout manager with specified margins
        this.setLayout(new BorderLayout(15, 15));
        resize(insets().left + insets().right + 261, insets().top + insets().bottom
```

```java
//** standard prefs for dialogs
setFont(new Font("TimesRoman", Font.PLAIN, 12));
setForeground(Color.black);
setBackground(Keywordcolordefault);

//** change text color to red
this.txcolor = txcolor;
if (txcolor == 2) setForeground(Color.red);
else if (txcolor == 3) setForeground(Color.yellow);

//** Create the message component and add it to the window
label = new Label(message);
this.add("Center", label);

//** Create an Okay button in a Panel; add the Panel to the window
//** Use a FlowLayout to center the button and give it margins.
button = new Button("Okay");
Panel p = new Panel();
p.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 15));
p.add(button);
add("South", p);

//** Resize the window to the preferred size of its components
this.pack();
}

public MessageBox(int txcolor,
                  help parent,
                  String title,
                  String message,
                  Color Keywordcolordefault) {

//** Create a dialog with the specified title
super(parent, title, false);
setResizable(false);

//** Create and use a BorderLayout manager with specified margins
this.setLayout(new BorderLayout(15, 15));
resize(insets().left + insets().right + 261, insets().top + insets().bottom + 72);

//** standard prefs for dialogs
setFont(new Font("TimesRoman", Font.PLAIN, 12));
setForeground(Color.black);
setBackground(Keywordcolordefault);

//** change text color to red
this.txcolor = txcolor;
if (txcolor == 2) setForeground(Color.red);
else if (txcolor == 3) setForeground(Color.yellow);

//** Create the message component and add it to the window
label = new Label(message);
this.add("Center", label);

//** Create an Okay button in a Panel; add the Panel to the window
//** Use a FlowLayout to center the button and give it margins.
button = new Button("Okay");
Panel p = new Panel();
p.setLayout(new FlowLayout(FlowLayout.CENTER, 15, 15));
p.add(button);
add("South", p);
```

```java
//** Resize the window to the preferred size of its components
this.pack();
}

/*******************************************************
* Override show placing in center of frame.
*******************************************************/
public synchronized void show() {

    Rectangle bounds = getParent().bounds();
    Rectangle abounds = bounds();
    move(bounds.x + (bounds.width - abounds.width)/2,
         bounds.y + (bounds.height - abounds.height)/2);

    super.show();
}

/*******************************************************
* Pop down the window when the button is clicked.
*******************************************************/
public boolean action(Event e, Object arg)
{

    if (e.target == button) {
        if (txcolor == 2) setForeground(Color.black);
        this.hide();
        this.dispose();
        return true;
    }

    else return false;

}

/*******************************************************
* When the window gets the keyboard focus, give it to the button.
* This allows keyboard shortcuts to pop down the dialog.
*******************************************************/
public boolean gotFocus(Event e, Object arg) {

    button.requestFocus();
    return true;

}

} //** end of MessageBox class

/*******************************************************
* KeyPressManagerPanel class
*
* Remarks: KeyPressManagerPanel is a Panel extension which provides for
* tabbing between components and supporting certain key accelerations
* by posting events provided by the user. Can be used directly or extended.
* When extending from BaseTabbedPanel be sure to super()
* during construction and to super.handleEvent(evt) from
* handleEvent if you override it.
* @version
* @author
*******************************************************/
```

```java
class KeyPressManagerPanel extends Panel
{

/* import java.awt.Button;
import java.awt.Component;
import java.awt.Container;
import java.awt.Event;
import java.awt.Label;
import java.awt.Panel;
import java.awt.TextComponent;
import java.util.Vector;
*/

//------------------------------
// constants
//------------------------------

    public static final int PLAIN = 0;
    public static final int SHIFT = Event.SHIFT_MASK;
    public static final int CTRL = Event.CTRL_MASK;


//------------------------------
// class variables
//------------------------------


//------------------------------
// member variables
//------------------------------

    Vector tabbed;
    Button defaultButton;
    Button cancelButton;
    Event defaultEvent;
    Event cancelEvent;
    Event fKeyEvents[];
    Container defaultDeliver;
    Container cancelDeliver;
    Container fKeyDeliver[];
    boolean bDefaultSetFocus;
    boolean bCancelSetFocus;
    boolean bTabHack;
    boolean bAutoTab;
    Event eventLostFocus;


//------------------------------
// constructors
//------------------------------

/**
 * Constructs a Panel which handles key press events.
 */
public KeyPressManagerPanel()
{
    fKeyEvents  = new Event[36];
    fKeyDeliver = new Container[36];
    bAutoTab    = true;
    resetKeyManager();
}
```

```java
//------------------------------
// accessor methods
//------------------------------

/**
 * Sets the automatic tab state.
 * @param bNewTabState new automatic tab state
 * @see #getAutoTabState
 */
public void setAutoTabState(boolean bNewTabState)
{
    bAutoTab = bNewTabState;
}

/**
 * Gets the current automatic tab state.
 * @return boolean - current automatic tab state value
 * @see #setAutoTabState
 */
public boolean getAutoTabState()
{
    return bAutoTab;
}

//------------------------------
// event methods
//------------------------------

public synchronized boolean handleEvent(Event evt)
{
    switch (evt.id)
    {
        case Event.KEY_ACTION:
        {
            int index = evt.key - Event.F1;

            switch (evt.modifiers)
            {
                case PLAIN:
                {
                    break;

                case SHIFT:
                {
                    index += 12;
                    break;

                case CTRL:
                {
                    index += 24;
                    break;

                default:
                {
                    index = -1;
                    break;
                }
            }

            if ((index > -1) && (index < 36))
            {
                if (fKeyEvents[index] != null)
```

```
r(index));
                    deliverEventTo(fKeyEvents[index], fKeyDelive
ist
                    )

                    return true;

                    )
            break;

            case Event.KEY_PRESS:
            (
                bTabHack = false;

                if (keyPressed(evt))
                (
                    return true;
                )
            break;

            case Event.LOST_FOCUS:
            (
                if (evt.target instanceof TextComponent)
                (
                    eventLostFocus = evt;
                    bTabHack = true;
                )
            break;

            case Event.KEY_RELEASE:
            (
                if (bTabHack)
                (
                    bTabHack = false;
                    eventLostFocus.key = evt.key;
                    eventLostFocus.modifiers = evt.modifiers;

                    if (keyPressed(eventLostFocus))
                    (
                        return true;
                    )
                )
            break;
            )

            return super.handleEvent(evt);

boolean keyPressed(Event evt)
(
    switch (evt.key)
    (
        case 9: // TAB:
        (
            if (evt.target instanceof Component)
            (
                return doTab((Component)evt.target, evt.modifiers);
            )

            return doTab((Component) this, evt.modifiers); // component not in l
```

```
        case 10: // ENTER:
        (
            if (defaultButton != null)
            (
                if (defaultButton.isEnabled())
                (
                    if (bDefaultSetFocus)
                    (
                        defaultButton.requestFocus(
                    )

                    deliverEventTo(defaultEvent, defaul

                    return true;
                )
            )
        )
        break;

        case 27: // ESC:
        (
            if (cancelButton != null)
            (
                if (cancelButton.isEnabled())
                (
                    if (bCancelSetFocus)
                    (
                        cancelButton.requestFocus()
                    )

                    deliverEventTo(cancelEvent, cancelE

                    return true;
                )
            )
        )
        break;
    )

    return false;
)

//------------------------------
// class methods
//------------------------------

//------------------------------
// memeber methods
//------------------------------

/**
 * Resets all KeyPressManager associations.
 */
public void resetKeyManager()
(
    tabbed         = new Vector();
    defaultButton  = null;
```

# QuestionApplet.java

```java
       cancelButton     = null;
       defaultEvent     = null;
       cancelEvent      = null;
       defaultDeliver   = null;
       cancelDeliver    = null;
       bDefaultSetFocus = false;
       bCancelSetFocus  = false;

       for (int x = 0; x < 36; x++)
       {
           fKeyEvents[x]  = null;
           fKeyDeliver[x] = null;
       }
   }

   /**
   * Sets the components as the next tab stop in the list
   * components.
   * @param component the Component
   * @return Component - returns added component
   */
   public Component add(Component component)
   {
   if(bAutoTab)
   {
       if (!(component instanceof Label))
       {
           setTabStop(component);
       }
   }

       return super.add(component);
   }

   /**
   * Removes Enter/Return key association with current default
   * button/event
   * @see #setDefaultButton
   */
   public void removeDefaultButton()
   {
       defaultButton    = null;
       defaultEvent     = null;
       defaultDeliver   = null;
       bDefaultSetFocus = false;
   }

   /**
   * Sets the components as the next tab stop in the list
   * components.
   * @param component the Component
   */
   public void setTabStop(Component component)
   {
       if (component != this)
       {
           tabbed.addElement(component);
       }
   }

   /**
   * Sets the button to press when the Enter or Return key
   * is pressed.
   * @param button the button to set as default
   * @see #removeDefaultButton
   */
   public void setDefaultButton(Button button)
   {
       setDefaultButton(button, new Event(button, Event.ACTION_EVENT, null
true);
   }

   /**
   * Associates a button and event with the Enter or Return
   * key press.
   * @param button the button to set as default
   * @param evt the event to delivered in response
   * @param deliverTo the container to deliver the event to
   * @param bSetFocus whether to set focus to the button before
   * delivering event
   * @see #removeDefaultButton
   */
   public void setDefaultButton(Button button, Event evt, Container deliverTo,
bSetFocus)
   {
       defaultButton    = button;
       defaultEvent     = evt;
       defaultDeliver   = deliverTo;
       bDefaultSetFocus = bSetFocus;
       button.requestFocus();
   }

   /**
   * Sets the button to press when the Escape key is pressed.
   * @param button the button to set as Cancel
   */
   public void setCancelButton(Button button)
   {
       setCancelButton(button, new Event(button, Event.ACTION_EVENT, null)
rue);
   }

   /**
   * Associates a button and event with the Escape key press.
   * @param button the button to set as Cancel
   * @param evt the event to delivered in response
   * @param deliverTo the container to deliver the event to
   * @param bSetFocus whether to set focus to the button before
   * delivering event
   */
   public void setCancelButton(Button button, Event evt, Container deliverTo,
SetFocus)
   {
       cancelButton    = button;
       cancelEvent     = evt;
       bCancelSetFocus = bSetFocus;
       cancelDeliver   = deliverTo;
   }

   /**
   * Removes Escape key association with current Cancel button/event
   */
   public void removeCancelButton()
   {
       cancelButton    = null;
```

```java
    cancelEvent     = null;
    cancelDeliver   = null;
    bCancelSetFocus = false;
}

/**
 * Associates an event with a Function key press
 * @param fKey the Event.F1 - Event.F12 constant
 * @param evt the event to delivered in response
 * @param deliverTo the container to deliver the event to
 */
public void setFKeyEvent(int fKey, Event evt, Container deliverTo)
{
    setFKeyEvent(fKey, PLAIN, evt, deliverTo);
}

/**
 * Removes association of an event with a Function key press
 * @param fKey the Event.F1 - Event.F12 constant
 */
public void removeFKeyEvent(int fKey)
{
    removeFKeyEvent(fKey, PLAIN);
}

/**
 * Associates an event with a Function key press
 * @param fKey the Event.F1 - Event.F12 constant
 * @param modifier PLAIN, SHIFT, or CTRL modifier
 * @param evt the event to delivered in response
 * @param deliverTo the container to deliver the event to
 */
public void setFKeyEvent(int fKey, int modifier, Event evt, Container deliverTo)
{
    int index = fKey - Event.F1;

    if ((index < 0) || (index > 11))
    {
        return;
    }

    switch (modifier)
    {
        case PLAIN:
        {
            break;
        }
        case SHIFT:
        {
            index += 12;
            break;
        }
        case CTRL:
        {
            index += 24;
            break;
        }
    }

    fKeyEvents[index] = evt;
    fKeyDeliver[index] = deliverTo;
}

/**
 * Removes association of an event with a Function key press
 * @param fKey the Event.F1 - Event.F12 constant
 * @param modifier PLAIN, SHIFT, or CTRL modifier
 */
public void removeFKeyEvent(int fKey, int modifier)
{
    int index = fKey - Event.F1;

    if ((index < 0) || (index > 11))
    {
        return;
    }

    switch (modifier)
    {
        case PLAIN:
        {
            break;
        }
        case SHIFT:
        {
            index += 12;
            break;
        }
        case CTRL:
        {
            index += 24;
            break;
        }
    }

    fKeyEvents[index] = null;
    fKeyDeliver[index] = null;
}

void deliverEventTo(Event evt, Container deliverTo)
{
    if (deliverTo == null)
    {
        postEvent(evt);
    }
    else
    {
        deliverTo.postEvent(evt);
    }
}

boolean doTab(Component current, int tabModifiers)
{
    int sze = tabbed.size();

    if (sze > 0 && (tabModifiers == 0 || tabModifiers == Event.SHIFT_MA
    {
        Component tabTo = null;
        int idx      = tabbed.indexOf(current);
        int iCurrent = idx;

        if (idx == -1)
        {
            Component c = current;

            while (c != this && idx == -1)
```

```java
        idx = tabbed.indexOf(c);
        c = c.getParent();
    }

    if (idx == -1)
    {
        if (tabModifiers == 0)
        {
            doTab((Component) tabbed.lastElement(), tabModifiers
        }
        else
        {
            doTab((Component) tabbed.firstElement(), tabModifier
        }

        return true;
    }
    else
    {
        int tabFromIdx = idx;

        while (true)
        {
            if (tabModifiers == 0) // regular tab
            {
                if (++idx == sze)
                {
                    idx = 0;
                }
            }
            else
            {
                if (--idx == -1)
                {
                    idx = sze - 1;
                }
            }

            if (tabFromIdx == idx)  // looped through all items
            {                        and none were enabled
                break;
            }

            try
            {
                tabTo = (Component) tabbed.elementAt(idx);

                if (tabTo.isEnabled())
                {
                    tabTo.requestFocus();

                    if (tabTo instanceof TextComponent)
                    {
                        TextComponent tc = (TextComponent) tabTo;

                        tc.selectAll();
                    }
```

```java
        Component tabFrom = (Component) tabbed.elem
ent.LOST_FOCUS, null);
            Event eLostFocus = new Event(ta

        tabFrom.postEvent(eLostFocus);

            tabTo = (Component) tabbed.elementA
.GOT_FOCUS, null);
            Event eGotFocus = new Event(tab

        tabTo.postEvent(eGotFocus);

            return true;
        }
    catch (ArrayIndexOutOfBoundsException e)
    {
    }
    }
}
```

```java
/*********************************************************************
*  ScrollingPanel class
*
*  package symantec.itools.awt;
*
*  import java.awt.Panel;
*  import java.awt.Component;
*  import java.awt.Scrollbar;
*  import java.awt.Dimension;
*  import java.awt.Color;
*  import java.awt.Rectangle;
*  import java.awt.Graphics;
*  import java.awt.Event;
*  import java.awt.LayoutManager;
*  import symantec.itools.awt.shape.Rect;
*  import symantec.itools.awt.KeyPressManagerPanel;
*
*  This class forms a ScrollingPanel component.
*  @version 1.0, Nov 26, 1996
*  @author Symantec
*********************************************************************/
class ScrollingPanel
    extends KeyPressManagerPanel
{
    private Component spComponent;
    private int width;
    private int height;
    private int xCoord, yCoord;
    private Scrollbar VBar = null;
    private Scrollbar HBar = null;
    private boolean bVBarVisible;
    private boolean bHBarVisible;
    private boolean bCornerRectVisible;
    private boolean bOsFlag;
    private int vPageSize;
    private int hPageSize;
    private Dimension dimComponent;
    private int vGapWid;
```

```java
private int hGapHt;
private boolean bAllowShowVBar = true;
private boolean bAllowShowHBar = true;
private int scrollLineIncrement = 1;
private Rect cornerRect;

//-------------------------------------
// constructors
//-------------------------------------

/**
 * Constructs a default ScrollingPanel.
 * The panel is initialized with a null component, zero
 * minimum height and zero minimum width.
 */
public ScrollingPanel()
{
    this(null, 0, 0);
}

/**
 * Constructs a new ScrollingPanel initialized with the
 * specified component, minimum height and minimum width.
 * @param component the component (usually a Panel) to be
 * scrolled
 * @param minWidth value to be used for the minimumSize()
 * width of the ScrollingPanel
 * @param minHeight value to be used for the minimumSize()
 * height of the ScrollingPanel
 */
public ScrollingPanel(Component component, int minWidth, int minHeight)
{
    bOsFlag = System.getProperty("os.name").startsWith("S"); // SunOS, Solaris
    this.spComponent = component;
    this.width = minWidth;
    this.height = minHeight;
    xCoord = 0;
    yCoord = 0;
    bVBarVisible = false;
    bHBarVisible = false;
    bCornerRectVisible = false;
    vPageSize = 0;
    hPageSize = 0;
    vGapWid = 6;
    hGapHt = 6;
    dimComponent = new Dimension(0,0);

    VBar = new Scrollbar();
    VBar.setBackground(Color.lightGray);

    HBar = new Scrollbar(Scrollbar.HORIZONTAL);
    HBar.setBackground(Color.lightGray);

    cornerRect = new Rect();
    cornerRect.setForeground(Color.lightGray);
    cornerRect.setFillColor(Color.lightGray);
    cornerRect.setFillMode(true);

    setLayout(null);
    super.add(VBar, -1);
    super.add(HBar, -1);
    super.add(cornerRect, -1);

    VBar.hide();
    HBar.hide();
    cornerRect.hide();

    if (spComponent != null)
    {
        super.add(spComponent, -1);
        placeComponents();
    }
}

/**
 * Set the value to be used for the minimumSize() width
 * of the ScrollingPanel
 * @param minWidth value to be used for the minimumSize()
 * width of the ScrollingPanel
 * @see #getMinimumWidth
 */
public void setMinimumWidth(int minWidth)
{
    this.width = minWidth;
}

/**
 * Get the current value used for the minimumSize()
 * width of the ScrollingPanel
 * @return int - current width value
 */
public int getMinimumWidth()
{
    return this.width;
}

/**
 * Set the value to be used for the minimumSize()
 * height of the ScrollingPanel
 * @param minHeight value to be used for the minimumSize()
 * height of the ScrollingPanel
 * @see #getMinimumHeight
 */
public void setMinimumHeight(int minHeight)
{
    this.height = minHeight;
}

/**
 * Get the value used for the minimumSize() height
 * of the ScrollingPanel
 * @return int - current height value
 * @see #setMinimumHeight
 */
public int getMinimumHeight()
{
    return this.height;
}

/**
 * Set the vertical gap amount between the container
 * in ScrollingPanel and the vertical scroll bar.
 * @param gapPixels size of vertical gap in pixels
 * @see #getVerticalGap
 */
public void setVerticalGap(int gapPixels)
```

```java
        vGapWid = gapPixels;
        invalidate();
    }

    /**
     * Get the current vertical gap amount between the
     * container in ScrollingPanel and the vertical scroll
     * bar.
     * @return int - size of vertical gap in pixels
     * @see #setVerticalGap
     */
    public int getVerticalGap()
    {
        return vGapWid;
    }

    /**
     * Set the horizontal gap amount between the container
     * in ScrollingPanel and the horizontal scroll bar.
     * @param gapPixels size of horizontal gap in pixels
     * @see #getHorizontalGap
     */
    public void setHorizontalGap(int gapPixels)
    {
        hGapHt = gapPixels;
        invalidate();
    }

    /**
     * Get the current horizontal gap amount between the
     * container in ScrollingPanel and horizontal scroll
     * bar.
     * @return int - size of horizontal gap in pixels
     * @see #setHorizontalGap
     */
    public int getHorizontalGap()
    {
        return hGapHt;
    }

    /**
     * Set whether or not the vertical scrollbar should be
     * made visible when necessary or should never be made
     * visible.
     * @param cond if true, show the scrollbar when necessary;
     * if false, never show the scrollbar
     * @see #getShowVerticalScroll
     */
    public void setShowVerticalScroll(boolean cond)
    {
        if (bAllowShowVBar != cond)
        {
            bAllowShowVBar = cond;
            invalidate();
        }
    }

    /**
     * Get the current vertical scrollbar visibility flag.
     * @return boolean - if true, show the scrollbar when
     * necessary; if false, never show the scrollbar
     * @see #setShowVerticalScroll
     */
    public boolean getShowVerticalScroll()
    {
        return bAllowShowVBar;
    }

    /**
     * Set whether or not the horizontal scrollbar should be
     * made visible when necessary or should never be made
     * visible.
     * @param cond if true, show the scrollbar when necessary;
     * if false, never show the scrollbar
     * @see #getShowHorizontalScroll
     */
    public void setShowHorizontalScroll(boolean cond)
    {
        if (bAllowShowHBar != cond)
        {
            bAllowShowHBar = cond;
            invalidate();
        }
    }

    /**
     * Get the current horizontal scrollbar visibility flag.
     * @return boolean - if true, show the scrollbar when
     * necessary; if false, never show the scrollbar
     * @see #setShowHorizontalScroll
     */
    public boolean getShowHorizontalScroll()
    {
        return bAllowShowHBar;
    }

    /**
     * Set the pixel increment to scroll for every scrollbar
     * arrow press.
     * @param scrollLineIncrement the pixel value to scroll,
     * default is one pixel
     * @see #getScrollLineIncrement
     */
    public void setScrollLineIncrement(int scrollLineIncrement)
    {
        this.scrollLineIncrement = scrollLineIncrement;
    }

    /**
     * Get the current pixel scroll increment.
     * @return the current pixel value amount to scroll
     * @see #setScrollLineIncrement
     */
    public int getScrollLineIncrement()
    {
        return scrollLineIncrement;
    }

    /**
     * Set the component in the ScrollingPanel.
     * @param comp the new component to be in the ScrollingPanel
     * @see #getComponent
     */
    public void setComponent(Component comp)
    {
```

```java
if (this.spComponent != null)
{
    super.remove(this.spComponent);
}

this.spComponent = comp;
super.add(spComponent, -1);
invalidate();
}

/**
 * Get the current component in the ScrollingPanel
 * @return the current component in the ScrollingPanel
 * @see #setComponent
 */
public Component getComponent()
{
    return this.spComponent;
}

public boolean handleEvent(Event evt)
{
    switch (evt.id)
    {
        case Event.SCROLL_LINE_UP:
            if (evt.target == HBar)
            {
                scrollLeft();
            }
            else
            {
                scrollUp();
            }
            return true;

        case Event.SCROLL_LINE_DOWN:
            if (evt.target == HBar)
            {
                scrollRight();
            }
            else
            {
                scrollDown();
            }
            return true;

        case Event.SCROLL_PAGE_UP:
            if (evt.target == HBar)
            {
                scrollPageLeft();
            }
            else
            {
                scrollPageUp();
            }
            return true;

        case Event.SCROLL_PAGE_DOWN:
            if (evt.target == HBar)
            {
                scrollPageRight();
            }
            else
            {
                scrollPageDown();
            }
            return true;

        case Event.SCROLL_ABSOLUTE:
            if (evt.target == HBar)
            {
                scrollHorizontalAbsolute(((Integer)evt.arg).intValue());
            }
            else
            {
                scrollVerticalAbsolute(((Integer)evt.arg).intValue());
            }
            return true;
    }

    return super.handleEvent(evt);
}

public void update(Graphics g)
{
    paint(g);
}

public void paint(Graphics g)
{
    if (spComponent == null)
    {
        return;
    }

    placeComponents();
}

/**
 * Scroll one pixel up.
 * @see #scrollDown
 * @see #scrollLeft
 * @see #scrollRight
 */
public void scrollUp()
{
    yCoord += scrollLineIncrement;

    if (yCoord > 0)
    {
```

```java
        yCoord = 0;
    }

    VBar.setValue(-yCoord);
    repaint();
}

/**
 * Scroll one pixel left.
 * @see #scrollRight
 * @see #scrollUp
 * @see #scrollDown
 */
public void scrollLeft()
{
    xCoord += scrollLineIncrement;

    if (xCoord > 0)
    {
        xCoord = 0;
    }

    HBar.setValue(-xCoord);
    repaint();
}

/**
 * Scroll one pixel down.
 * @see #scrollUp
 * @see #scrollLeft
 * @see #scrollRight
 */
public void scrollDown()
{
    yCoord -= scrollLineIncrement;

    if ( (-yCoord) > VBar.getMaximum() )
    {
        yCoord = -VBar.getMaximum();
    }

    VBar.setValue(-yCoord);
    repaint();
}

/**
 * Scroll one pixel right.
 * @see #scrollLeft
 * @see #scrollUp
 * @see #scrollDown
 */
public void scrollRight()
{
    xCoord -= scrollLineIncrement;

    if ( (-xCoord) > HBar.getMaximum() )
    {
        xCoord = -HBar.getMaximum();
    }

    HBar.setValue(-xCoord);
    repaint();
}
```

```java
/**
 * Scroll one "page" up.
 * @see #scrollPageDown
 * @see #scrollPageLeft
 * @see #scrollPageRight
 */
public void scrollPageUp()
{
    yCoord += vPageSize;

    if (yCoord > 0)
    {
        yCoord = 0;
    }

    VBar.setValue(-yCoord);
    repaint();
}

/**
 * Scroll one "page" left.
 * @see #scrollPageRight
 * @see #scrollPageUp
 * @see #scrollPageDown
 */
public void scrollPageLeft()
{
    xCoord += hPageSize;

    if (xCoord > 0)
    {
        xCoord = 0;
    }

    HBar.setValue(-xCoord);
    repaint();
}

/**
 * Scroll one "page" down.
 * @see #scrollPageUp
 * @see #scrollPageLeft
 * @see #scrollPageRight
 */
public void scrollPageDown()
{
    yCoord -= vPageSize;

    if ( (-yCoord) > VBar.getMaximum() )
    {
        yCoord = -VBar.getMaximum();
    }

    VBar.setValue(-yCoord);
    repaint();
}

/**
 * Scroll one "page" right.
 * @see #scrollPageLeft
 * @see #scrollPageUp
 * @see #scrollPageDown
```

```java
    */
    public void scrollPageRight()
    {
        xCoord -= hPageSize;

        if ( (-xCoord) > HBar.getMaximum())
        {
            xCoord = -HBar.getMaximum();
        }

        HBar.setValue(-xCoord);
        repaint();
    }

    /**
     * Scroll to an absolute vertical position.
     * @param int the pixel position to scroll to
     * @see #scrollHorizontalAbsolute
     */
    public void scrollVerticalAbsolute(int position)
    {
        yCoord = -position;

        if (yCoord > 0)
        {
            yCoord = 0;
        }
        else if ( (-yCoord) > VBar.getMaximum())
        {
            yCoord = -VBar.getMaximum();
        }

        VBar.setValue(-yCoord);
        repaint();
    }

    /**
     * Scroll to an absolute horizontal position.
     * @param int the pixel position to scroll to
     * @see #scrollVerticalAbsolute
     */
    public void scrollHorizontalAbsolute(int position)
    {
        xCoord = -position;

        if (xCoord > 0)
        {
            xCoord = 0;
        }
        else if ( (-xCoord) > HBar.getMaximum())
        {
            xCoord = -HBar.getMaximum();
        }

        HBar.setValue(-xCoord);
        repaint();
    }

    public Dimension preferredSize()
    {
        Dimension s = size();
        Dimension m = minimumSize();
        return new Dimension(Math.max(s.width, m.width), Math.max(s.height, m.height));
    }


    public Dimension minimumSize()
    {
        return new Dimension(width, height);
    }

    public Component add(Component comp)
    {
        if (this.spComponent != null)
        {
            super.remove(this.spComponent);
        }

        this.spComponent = comp;
        super.add(spComponent, -1);
        repaint();

        return comp;
    }

    public synchronized Component add(Component comp, int pos)
    {
        return add(comp);
    }

    public synchronized Component add(String name, Component comp)
    {
        return add(comp);
    }

    public synchronized void remove(Component comp)
    {
        if (comp == VBar || comp == HBar)
        {
            return;
        }

        super.remove(comp);

        if (comp == spComponent)
        {
            spComponent = null;
        }
    }

    public synchronized void removeAll()
    {
        super.removeAll();
        super.add(VBar, -1);
        super.add(HBar, -1);
        super.add(cornerRect, -1);
        spComponent = null;
    }

    public void setLayout(LayoutManager mgr)
    {
    }

    public synchronized void reshape(int x, int y, int width, int height)
    {
        repaint();
```

```java
    super.reshape(x, y, width, height);
}

void placeComponents()
{
    boolean bShowV, bShowH;
    int barSize = 0;
    int vWid = 0;
    int hHt = 0;
    dimComponent = spComponent.size();
    Rectangle rect = bounds();

    barSize = bOsFlag ? 17 : 15;

    if (bAllowShowHBar && dimComponent.width > rect.width)
    {
        bShowH = true;
        hHt = barSize;
    }
    else
    {
        bShowH = false;
        hHt = 0;
    }

    if (bAllowShowVBar && dimComponent.height > (rect.height-hHt))
    {
        bShowV = true;
        vWid = barSize;

        if (!bShowH)
        {
            if (dimComponent.width > (rect.width-vWid))
            {
                bShowH = true;
                hHt = barSize;
            }
        }
    }
    else
    {
        bShowV = false;
        vWid = 0;
    }

    hPageSize = rect.width - vWid - vGapWid;
    vPageSize = rect.height - hHt - hGapHt;

    if (bShowV)
    {
        VBar.reshape(rect.width-barSize, 0, barSize, rect.height-hHt);
        VBar.setValues(-yCoord, vPageSize, 0, dimComponent.height - vPageSize);
        VBar.setPageIncrement(vPageSize);

        if (!bVBarVisible)
        {
            bVBarVisible = true;
            VBar.show();
        }
    }
    else
    {
        if (bVBarVisible)
        {
            bVBarVisible = false;
            VBar.hide();
        }

        yCoord = 0;
    }

    if (bShowH)
    {
        HBar.reshape(0, rect.height-barSize, rect.width-vWid, barSize);
        HBar.setValues(-xCoord, hPageSize, 0, dimComponent.width - hPageSize);
        HBar.setPageIncrement(hPageSize);

        if (!bHBarVisible)
        {
            bHBarVisible = true;
            HBar.show();
        }
    }
    else
    {
        if (bHBarVisible)
        {
            bHBarVisible = false;
            HBar.hide();
        }

        xCoord = 0;
    }

    if (bHBarVisible && bVBarVisible)
    {
        int x = rect.width - vWid;
        int y = rect.height - hHt;
        int w = rect.width - x + 1;
        int h = rect.height - y + 1;
        cornerRect.reshape(x, y, w, h);

        if (!bCornerRectVisible)
        {
            bCornerRectVisible = true;
            cornerRect.show();
        }
    }
    else
    {
        if (bCornerRectVisible)
        {
            bCornerRectVisible = false;
            cornerRect.hide();
        }
    }

    spComponent.move(xCoord, yCoord);
    spComponent.validate();
}

/*****************************************************
 * Rect class
 * package symantec.itools.awt.shape;
 *
```

```
import java.awt.Graphics;
import java.awt.Color;

/*
 * This class forms the Rectangle shape component.
 * @see symantec.itools.awt.shape.HorizontalLine
 * @see symantec.itools.awt.shape.Square
 * @see symantec.itools.awt.shape.VerticalLine
 * @version 1.0, Nov 26, 1996
 * @author Symantec
 *********************************************************/
class Rect extends Shape
{
    /**
     * Constructs a default Rectangle.
     */
    public Rect()
    {
    }

    /**
     * Paints the rectangle.
     */
    public void paint(Graphics g)
    {
        g.clipRect(0, 0, width, height);

        int w = width - 1, h = height - 1;

        switch (style) {

        case BEVEL_LINE :
        default :
            if (fill) {
                g.setColor(fillColor);
                g.fillRect(0, 0, w, h);
            }
            else
                g.drawRect(0, 0, w, h);
            break;

        case BEVEL_LOWERED :
            g.setColor(Color.gray);
            g.drawLine(0, h, 0, 0);
            g.drawLine(0, 0, w, 0);

            g.setColor(Color.white);
            g.drawLine(w, 0, w, h);
            g.drawLine(w, h, 0, h);

            if (fill) {
                g.setColor(fillColor);
                g.fillRect(1, 1, w - 1, h - 1);
            }
            break;

        case BEVEL_RAISED :
            g.setColor(Color.white);
            g.drawLine(0, h, 0, 0);
            g.drawLine(0, 0, w, 0);

            g.setColor(Color.gray);
            g.drawLine(w, 0, w, h);
            g.drawLine(w, h, 0, h);

            if (fill) {
                g.setColor(fillColor);
                g.fillRect(1, 1, w - 1, h - 1);
            }
            break;
        }
    }
}

/****************************************************************************
 * package symantec.itools.awt.shape;

import java.awt.Canvas;
import java.awt.Dimension;
import java.awt.Color;
import symantec.itools.awt.BevelStyle;

/*
 * This is the parent Shape class for the various
 * shape components.
 * @see symantec.itools.awt.shape.Ellipse
 * @see symantec.itools.awt.shape.Rectangle
 * @version 1.0, Nov 26, 1996
 * @author Symantec
 ****************************************************************************/
abstract class Shape
    extends     Canvas
    implements BevelStyle
{
    protected int width;
    protected int height;
    protected int style;
    protected boolean fill;
    protected Color fillColor;

    protected Shape()
    {
        style = BEVEL_LINE;
    }

    /**
     * Sets the border style of the shape.
     * @see #getStyle
     */
    public void setBevelStyle(int s)
    {
        style = s;
        repaint();
    }

    /**
     * Returns the current style of the shape.
     * @see #setStyle
     */
    public int getBevelStyle()
    {
        return style;
    }

    /**
     * Sets the fill mode of the shape.
     * @see #getFillMode
```

```java
    */
    public void setFillMode(boolean f)
    {

        fill = f;
        repaint();

    }

    /**
     * Returns the current fill mode of the shape.
     * @see #setFillMode
     */
    public boolean getFillMode()
    {

        return fill;

    }

    /**
     * Sets the fill color of the shape.
     * @see #getFillColor
     */
    public void setFillColor(Color color)
    {

        fillColor = color;
        repaint();

    }

    /**
     * Returns the current fill color of the shape.
     * @see #setFillColor
     */
    public Color getFillColor()
    {

        return fillColor;

    }

    /**
     * Reshapes the shape.
     */
    public void reshape(int x, int y, int width, int height)
    {

        this.width  = width;
        this.height = height;

        super.reshape(x, y, width, height);

    }

    /**
     * Returns the current "preferred size" of the shape.
     */
    public Dimension preferredSize()
    {

        return new Dimension(50, 50);

    }


/***************************************************************************
*package symantec.itools.awt;
*
***************************************************************************/
interface BevelStyle
{

    //-----------------------------------------------------------------------
    // constants
    //-----------------------------------------------------------------------

    /**
     * Defines the "lowered" shape border style.
     */
    public static final int BEVEL_LOWERED = 0;

    /**
     * Defines the "raised" shape border style.
     */
    public static final int BEVEL_RAISED = 1;

    /**
     * Defines the "line" shape border style.
     */
    public static final int BEVEL_LINE = 2;

    /**
     * Defines the "line" shape border style.
     */
    public static final int BEVEL_NONE = 3;

    //-----------------------------------------------------------------------
    // methods
    //-----------------------------------------------------------------------

    public void setBevelStyle(int style);
    public int getBevelStyle();


/***************************************************************************
* catalog help
*
* Remarks: This class calls the catalog module
*
***************************************************************************/
class help extends Frame {

    //** declarations
    Label label1;
    Color Keywordcolordefault = Color.red;
    Label showlabel;
    Choice choice1;
    Button printbutton;
    Button closebutton;
    Button helpbutton;
    Label keywordlabel;
    TextField searchtext;
    Button searchbutton;
    TextArea textarea;
    Scrollbar scrollbar1;
    String findText     = "";


    public help() {

        super("QuestionApplet Help - 1.0");

        setResizable(false);

        setLayout(null);
```

```java
        addNotify();
        resize(insets().left + insets().right + 529, insets().top + insets().bottom + 421);

        setCursor(Frame.HAND_CURSOR);

        closebutton=new Button("Close");
        add(closebutton);
        closebutton.reshape(insets().left + 400,insets().top + 10,78,26);

        keywordlabel=new Label("Key Word Search:");
        add(keywordlabel);
        keywordlabel.reshape(insets().left + 10,insets().top + 10,114,26);

        searchtext=new TextField(11);
        add(searchtext);
        searchtext.reshape(insets().left + 136,insets().top + 10,96,30);

        searchbutton=new Button("Search");
        add(searchbutton);
        searchbutton.reshape(insets().left + 250,insets().top + 10,78,26);

        textarea=new TextArea();
        add(textarea);
        textarea.reshape(insets().left + 10,insets().top + 70,486,312);

        textarea.setText("The NPS Question Applet\n"+
            "    Display the NPS Course Catalog\n"+
            "Short List\n"+
            "    Display the NPS Course Short List\n"+
            "Print\n"+
            "    Print the Course Planner\n"+
            "Grad Status\n"+
            "    Checks matrix for errors and omissions\n"+
            "Planner Matrix\n"+
            "    To enter a course on the matrix,\n"+
            "    highlight the desired cell and begin typing.\n"+
            "    Press return.\n"+
            "Track Select\n"+
            "    Choose a track to view its matrix\n");

        show();
    }

    public synchronized void show() {
        move(50, 50);
        super.show();
    }

    public boolean handleEvent(Event event) {

    if (event.id == Event.ACTION_EVENT && event.target == closebutton) {
        clickedclose();
        return true;

    else if (event.id == Event.ACTION_EVENT && event.target == searchbutton) {
        clickedsearch();
        return true;
    }
    else if (event.id == Event.WINDOW_DESTROY) {
        hide();       // hide the Frame
        dispose();    // tell windowing system to free resources

        return true;
    }
    return super.handleEvent(event);
}

/*********************************************************************
* Message()
*
* Remarks:  This method outputs Message and yes or no.
*********************************************************************/
public void Message(int txcolor,String title, String message) {
    MessageBox theMessageBox;
    theMessageBox = new MessageBox(txcolor,this,title,message,Color.yellow);
    theMessageBox.show();
} //end of Message

/*********************************************************
* showErrorDialog method
*
*********************************************************/
public void showErrorDialog(String message) {  .
    Message(2,"Error",message);
}

public void clickedclose() {
    handleEvent(new Event(this, Event.WINDOW_DESTROY, null));
}

public void clickedprint() {
    Message(3,"Printer Error","Printer Not Responding..Check Connection");
}

public void clickedsearch() {

    if (searchtext.getText().length() > 0) {
        setfindText(searchtext.getText());
        findText();
        textarea.requestFocus();
    }
}

/******************************************************
* setfindText method
*
******************************************************/
public void setfindText(String str) {
    findText = str;
}

/******************************************************
* findSelectedText method
*
******************************************************/
public void findSelectedText() {
    String sel = textarea.getSelectedText();
    if (sel.length() > 0) findText = sel;
    findText();
```

```
}

/*********************************************************
 * findText method
 *
 *********************************************************/
public void findText() {

    String s;
    int spoint, index;

    textarea.requestFocus();
    if (findText.length() == 0) return;

    s = textarea.getText();
    if (s.length() == 0) return;

    spoint = textarea.getSelectionEnd();
    index = s.substring(spoint).indexOf((findText);
    if (index >= 0) {
        int i = spoint + index;
        textarea.select(i, i + findText.length());
    }
}

} //** end of Help class
```

# INITIAL DISTRIBUTION LIST

1.  Defense Technical Information Center...............................................................2
    8725 John J. Kingman Road, Suite 0944
    Fort Belvoir, VA 22060

2.  Dudley Knox Library.....................................................................................2
    Naval Postgraduate School
    411 Dyer Road
    Monterey, CA 93940

3.  Dr. Ted Lewis, Chairman, Code CS...............................................................1
    Computer Science Department
    Naval Postgraduate School
    Monterey, CA 93940

4.  Dr. Dennis Volpano, Code CS/Vd................................................................2
    Computer Science Department
    Naval Postgraduate School
    Monterey, CA 93940

5.  Dr. Cynthia Irvine, Code CS/Ic...................................................................1
    Computer Science Department
    Naval Postgraduate School
    Monterey, CA 93940

6.  CPT. Wes Hester, Code SGC 1721 ..............................................................3
    Computer Science Department
    Naval Postgraduate School
    Monterey, CA 93940

7.  LT. Richard Moormann,...............................................................................3
    211 Pottery Road
    Washington, MO 63090

8.  Dr. Ray Griffin .............................................................................................1
    NETPDTC, Code N743
    6490 Saufley Field Road
    Pensacola, FL 32509-5239

9.  Don Brutzman, Code UW/B...........................................................................1
    Undersea Warfare
    Naval Postgraduate School
    Monterey, CA 93943